vmware | SPRING & CLOUD FOUNDRY 开发者大会

Spring 与 Cloud Foundry —— 天作之合

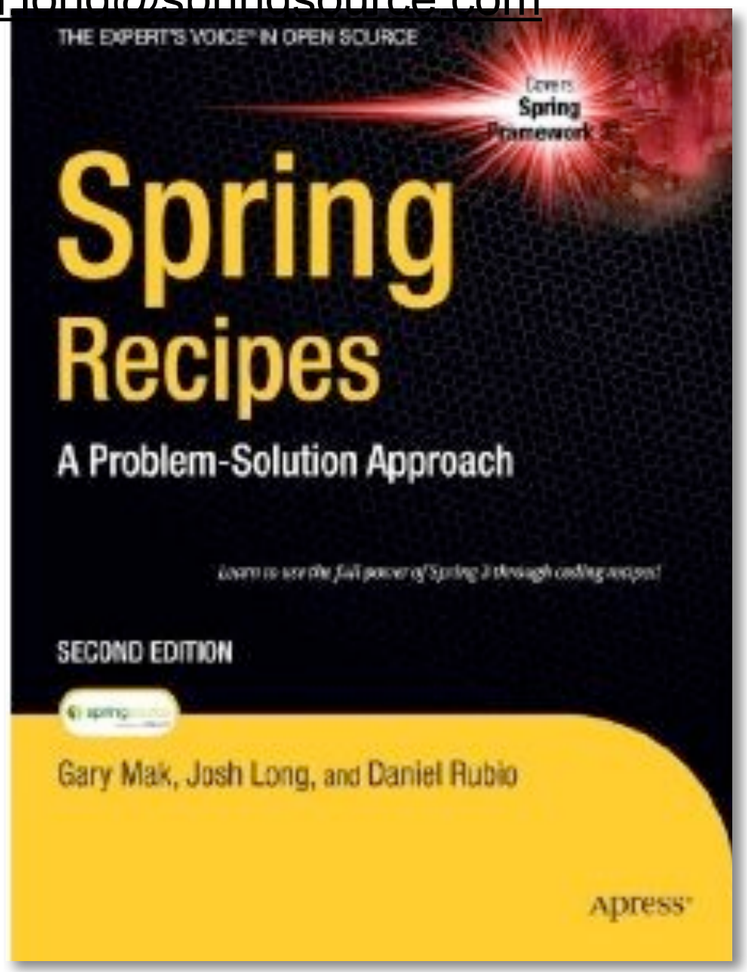
Josh Long, *VMWare 旗下 SpringSource 部门的* ***Spring*** *开发人员技术布道师*

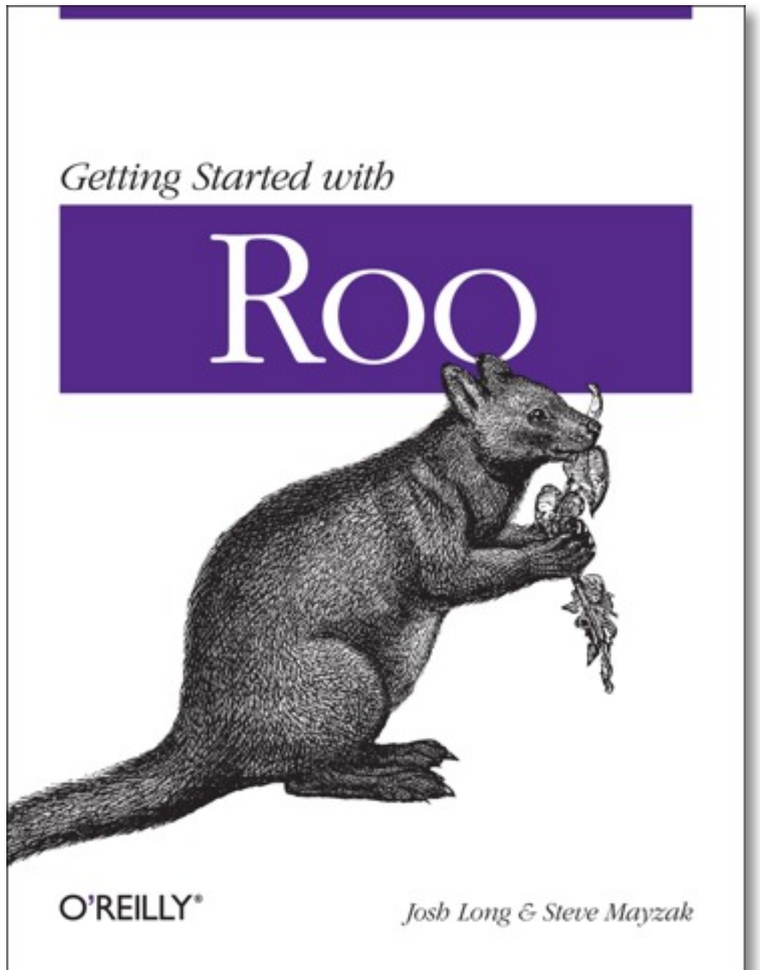
http://www.joshlong.com || @starbuxman || josh.long@springsource.com

Josh Long (龙之春) 介绍

Spring 开发人员技术布道师

twitter: @starbuxman
weibo: @springsource
josh.long@springsource.com





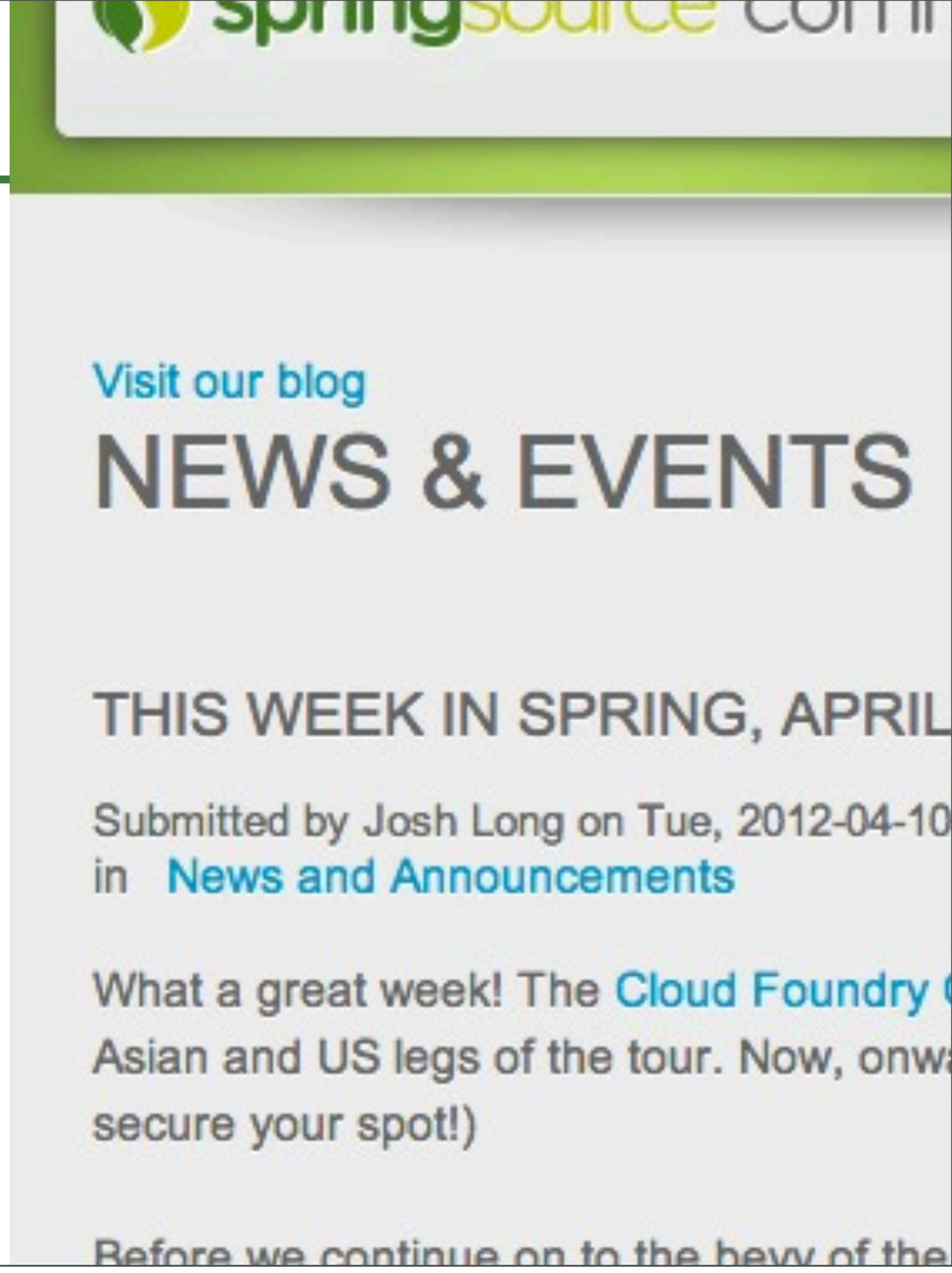
Josh Long (龙之春) 介绍

Spring 开发人员技术布道师

twitter: @starbuxman
josh.long@springsource.com

•参与贡献的项目：

- **Spring Integration**
- **Spring Batch**
- **Spring Hadoop**
- Activiti 工作流程引擎
- 使用 Akka Spring 模块



vmware | SPRING & CLOUD FOUNDRY
开发者大会

为什么选择 Cloud Foundry？

我们有令人惊叹的语言、工具和框架

创意 → **有效代码**

但是部署需要多长时间？

有效代码

已部署

但是部署需要多长时间？

有效代码

已部署

几小时？

但是部署需要多长时间？

有效代码

已部署

几小时？

几天？

但是部署需要多长时间？

有效代码

已部署

几小时？

几天？

几周？

但是部署需要多长时间？

有效代码

已部署

几小时？

几天？

几周？

几个月？

但是如果是开发人员...

设想一下，如果建筑师必须管理他们所设计的每座建筑，情况将如何？这就是开发团队在迁移至 Windows Azure 之前所面临的情况。

Duncan Mackenzie，2011 年 11 月 7 日
http://www.infoq.com/articles/Channel-9-Azure

云部署面临的难题

- 仍是针对应用程序的另一环境，以便使它们能够在
  - 不同的 Java 运行时中运行
  - 不同的数据库（有时并非关系数据库）中运行
- 动态环境 ⇒ 服务器名称不固定
- 弹性扩展 ⇒ 服务器来来往往
- 云应用程序必须与非云应用程序相集成

云部署面临的难题

- 仍是针对应用程序的另一环境，以便使它们能够在
  - 不同的 Java 运行时中运行
  - 不同的数据库（有时并非关系数据库）中运行
- 动态环境 ⇒ 服务器名称不固定
- 弹性扩展 ⇒ 服务器来来往往
- 云应用程序必须与非云应用程序相集成

Spring 框架已解决这些问题

vmware | SPRING & CLOUD FOUNDRY 开发者大会

为什么选择 Spring？

我们为什么在这里？

> 软件实体（类、模块、功能等）
> 应该可以扩展，而不允许修改。

-Bob Martin

我们为什么在这里？

千万不要
彻底改造车轮！

Spring 框架

- 企业 Java 的实际标准编程模型
- 两百多万位开发人员
- 快速发展
  - Spring 1.0 – 2004 年 3 月
  - Spring 2.0 – 2006 年 10 月
  - Spring 2.5 – 2007 年 12 月
  - Spring 3.0 – 2009 年 12 月
  - Spring 3.1 - 2011 年 12 月
  - Spring 3.2 - 2012 年 12 月
- 完全向后兼容性

Spring 的目标：

Web 层与	服务层	批处理	集成与	数据访问 / NoSQL /	移动	安全

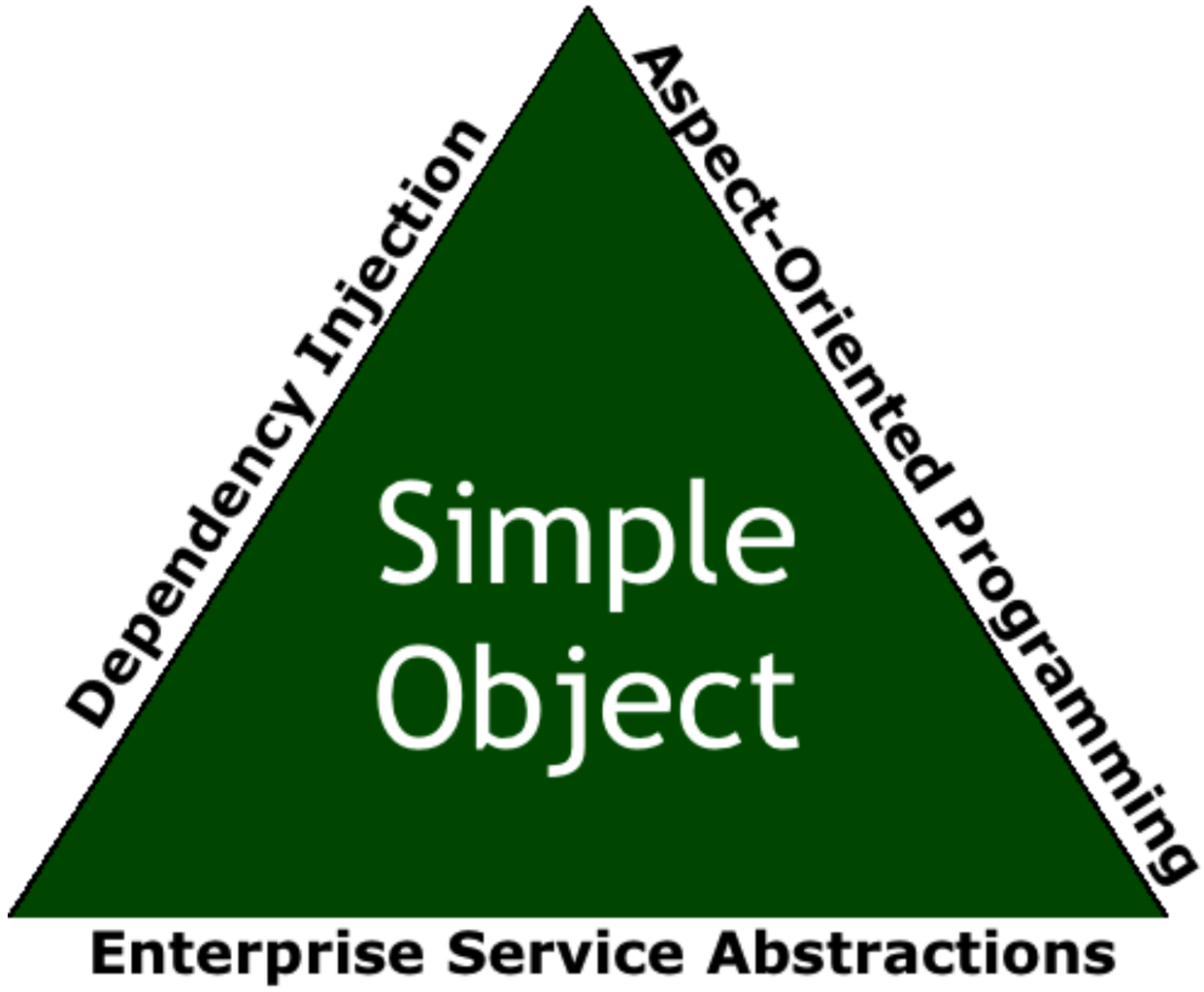
Spring 框架

云：	轻量型	传统
CloudFoundry Google App Engine Amazon BeanStalk	tc Server Tomcat Jettv	WebSphere JBoss AS WebLogic （基于旧版本、工具）

Spring 框架生态系统

框架	说明
Spring 框架	基础
Spring Security（又称为 Acegi）	提供身份验证、授权和实例级安全的可扩展框架
Spring Web Flow	用于构建多页流的出色 Web 框架
Spring Web Services	契约优先、以文档为中心的 SOAP 和 XML Web 服务
Spring Data	简化 SQL 与 NoSQL 存储的数据访问
Spring Batch	强大的批处理框架
Spring Integration	实施企业级集成模式
Spring Flex	支持 Adobe BlazeDS
Spring HATEOAS	支持符合 HATEOAS 模式的应用程序
Spring AMQP	用于连接 AMQP 消息代理的框架

并非所有方面都均衡


Dependency Injection
Aspect-Oriented Programming
Simple Object
Enterprise Service Abstractions

Spring 使构建应用程序变得简单...

告诉 Spring 您的目标

```
package org.springsource.examples.spring31.services;
...
@Configuration
@ComponentScan(“the.package.with.beans.in.it”)
public class ServicesConfiguration {

...

}
```

告诉 Spring 您的目标

```
public class Main {
  static public void main (String [] args) throws Throwable {
        ApplicationContext ac = new AnnotationConfigApplicationContext(
            org.springsource.examples.spring31.services.ServicesConfiguration.class);
        ...
  }
}
```

告诉 Spring 您的目标

```
package the.package.with.beans.in.it;

@Service
public class CustomerService {

    public Customer createCustomer(String firstName,
                                    String lastName,
                                    Date signupDate)    {

    }

    ...

}
```

我想要数据库访问...

```
package org.springsource.examples.spring31.services;
...
@Configuration
public class ServicesConfiguration {


    @Bean
    public DataSource dataSource() throws Exception {
        SimpleDriverDataSource simpleDriverDataSource =
          new SimpleDriverDataSource();
         ....
        return simpleDriverDataSource;
    }

}
```

我想要数据库访问... 具有 Hibernate 4 支持

```
package org.springsource.examples.spring31.services;
...
@Configuration
public class ServicesConfiguration {
    ...

    @Bean
    public SessionFactory sessionFactory() throws Exception {
        Properties props = new Properties();
        // ... show_sql, dialect, etc.
        return new LocalSessionFactoryBuilder( dataSource() )
                .addAnnotatedClasses(Customer.class)
                .addProperties(props)
                .buildSessionFactory();
    }

}
```

我想要数据库访问... 具有 Hibernate 4 支持

```
package the.package.with.beans.in.it;
...
@Service
public class CustomerService {

   @Inject
   private SessionFactory sessionFactory;

   public Customer createCustomer(String firstName,
                                  String lastName,
                                  Date signupDate)    {
      Customer customer = new Customer();
      customer.setFirstName(firstName);
      customer.setLastName(lastName);
      customer.setSignupDate(signupDate);

      sessionFactory.getCurrentSession().save(customer);
      return customer;
   }

   ...

}
```

我想要公布的事务管理...

```
package org.springsource.examples.spring31.services;
...
@Configuration
@EnableTransactionManagement
public class ServicesConfiguration {

   ...

   @Bean
   public PlatformTransactionManager transactionManager() throws Exception {
      return new HibernateTransactionManager(this.sessionFactory());
   }
}
```

我想要公布的事务管理...

```
package the.package.with.beans.in.it;
...
@Service
public class CustomerService {

    @Inject
    private SessionFactory sessionFactory;

    @Transactional
    public Customer createCustomer(String firstName,
                                   String lastName,
                                   Date signupDate)    {
        Customer customer = new Customer();
        customer.setFirstName(firstName);
        customer.setLastName(lastName);
        customer.setSignupDate(signupDate);

        sessionFactory.getCurrentSession().save(customer);
        return customer;
    }
    ...
}
```

我想要公布的缓存管理...

```
package org.springsource.examples.spring31.services;
...
@Configuration
@EnableTransactionManagement
@EnableCaching
public class ServicesConfiguration {
   ...

   @Bean
   public CacheManager cacheManager() {
      SimpleCacheManager scm = new SimpleCacheManager();
      Cache cache = new ConcurrentMapCache("customers");
      scm.setCaches(Arrays.asList(cache));
      return scm;
   }
}
```

我想要公布的缓存管理...

```
package the.package.with.beans.in.it;
...
@Service
public class CustomerService {

    @Inject
    private SessionFactory sessionFactory;

    @Transactional
    @Cacheable("customers")
    public Customer createCustomer(String firstName,
                                   String lastName,
                                   Date signupDate)    {
        Customer customer = new Customer();
        customer.setFirstName(firstName);
        customer.setLastName(lastName);
        customer.setSignupDate(signupDate);

        sessionFactory.getCurrentSession().save(customer);
        return customer;
    }
}
```

我想要基于 REST 的终端...

```
package org.springsource.examples.spring31.web;
..

@Controller
public class CustomerController {

    @Inject
    private CustomerService customerService;

    @RequestMapping(value = "/customer/{id}",
                    produces = MediaType.APPLICATION_JSON_VALUE)
    public @ResponseBody Customer customerById(@PathVariable("id") Integer id) {
        return customerService.getCustomerById(id);
    }
     ...
}
```

可移植性是王道

- Spring 始终专注于可移植性
  - Web 服务器和应用程序服务器之间的可移植性
  - 云之间的可移植性
  - 环境之间的可移植性

依赖项注入

- 组件不再负责查找它们的依赖项
- 相反，依赖项在实例化时传入
  - 构造函数注入
  - 资源库注入
  - 字段注入
- 也称为控制反转 – 不要呼叫我们（以获取依赖项），
  我们将呼叫您（即，传入依赖项）

vmware | SPRING & CLOUD FOUNDRY 开发者大会

构建针对 Cloud Foundry 的 Spring 应用程序

了解 CLOUD FOUNDRY 服务

Cloud Foundry：服务

- 服务是 Cloud Foundry 中可扩展的其中一个方面
  - 社区每天还会贡献更多服务！
- MySQL、Redis、MongoDB、RabbitMQ、PostgreSQL
- 服务可在不同应用程序间共享
- Cloud Foundry 通过托管在云控制器中的统一 API 将服务的配置抽象化
- 使用统一方法获取应用程序并将服务添加应用程序非常简单
  - Cassandra? COBOL / CICS, Oracle

Cloud Foundry：服务

- 利用服务
  - 设置服务无需成本
  - 服务能够实现价值

- 它们能够促进实现更出色的体系结构
  - 需要快速读写缓存？**Redis** 随时待命！
  - 需要存储长尾文档？不妨试试 **MongoDB**
  - 需要解除应用程序操作内容和操作时间的耦合？
    请使用消息传递和 **RabbitMQ**

Cloud Foundry 通过环境变量公开服务

$VCAP_SERVICES:
{"redis-2.2":[{"name":"redis_sample","label":"redis-2.2","plan":"free",
"tags":["redis","redis-2.2","key-value","nosql"],
"credentials":
{"hostname":"172.30.48.40",
"host":"172.30.48.40",
"port":5023,
"password":"8e9a901f-987d-4544-9a9e-ab0c143b5142",
"name":"de82c4bb-bd08-46c0-a850-af6534f71ca3"}
}],
"mongodb-1.8":[{"name":"mongodb-e7d29","label":"mongodb-1.8","plan":"free","tags”:
.....

Spring 是针对您服务的最佳工具包

- Spring Data
  - 支持高级 JPA、MongoDB、Redis 连接
- Spring AMQP、Spring Integration
  - 支持消息传递和事件驱动型体系结构
- Spring Core
  - 可通过 JPA、JDBC、JDO、Hibernate 等实现对 RDBMS 访问的最佳支持

自动重新配置：入门

- 无需更改任何代码即可将 Spring 应用程序部署到云
- Cloud Foundry 自动重新配置 Bean 定义以绑定到云服务
- 使用 Spring 和 Grails

自动重新配置：关系数据库

- 检测类型 javax.sql.DataSource 的 Bean
- 连接到 MySQL 或 PostgreSQL 服务
  - 指定驱动程序、URL、用户名、密码、验证查询
- 创建 Commons DBCP 或 Tomcat DataSource
- 替换现有 DataSource

```
<bean class = “...BasicDataSource” id=”dataSource”>
  <property name = “url” value = “jdbc:h2:mem”/>
  <property name =”password” value =””/>
  <property name = “username” value = “sa”/>
  <property name = “driverClass” value = “com.h2.Driver”/>
</bean>
```

```
import org.apache.commons.dbcp.BasicDataSource;
...
@Bean(destroyMethod = "close")
public BasicDataSource dataSource(){

  BasicDataSource  bds = new BasicDataSource();
  bds.setUrl( "jdbc:h2:mem");
  bds.setPassword("");
  bds.setUsername("sa");
  bds.setDriverClass( Driver.class);
  return bds;
}
```

自动重新配置：ORM

- 调整 Hibernate 方言
- 将 hibernate.dialect 属性更改为

  MySQLDialect (MyISAM) 或 PostgreSQLDialect

  – org.springframework.orm.jpa.AbstractEntityManagerFactoryBean
  – org.springframework.orm.hibernate3.AbstractSessionFactoryBean
  （Spring 2.5 和 3.0）

  – org.springframework.orm.hibernate3.SessionFactoryBuilderSupport
  (Spring 3.1)

```
@Bean
public LocalContainerEntityManagerFactoryBean entityManager(){
 LocalContainerEntityManagerFactoryBean lcem =
      new LocalContainerEntityManagerFactoryBean();
 lcem.setDataSource( dataSource() );
 return lcem;
}
```

自动重新配置：工作原理

- Cloud Foundry 在暂存期间会在您的应用程序上下文中安装 BeanFactoryPostProcessor
  - 将 jar 添加到您的应用程序
  - 修改 web.xml 以加载 BFPP
    - 将上下文文件添加到 contextConfigLocation
      - web-app context-param
      - Spring MVC DispatcherServlet init-param
- 添加 DataSource 重新配置所需的 PostgreSQL 和 MySQL 驱动程序

vmware | SPRING & CLOUD FOUNDRY 开发者大会

构建针对 Cloud Foundry 的 Spring 应用程序

明确地通过 java 绑定至服务

环境

- 提问
  - 您可以反省环境变量 (**System.getenv("..")**), 或...
  - 从 Java 导入 CloudFoundry 运行时！
    - （简单多了）

```
<dependency>
  <groupId>org.cloudfoundry</groupId>
   <artifactId>cloudfoundry-runtime</artifactId>
   <version>0.8.2</version>
 </dependency>
```

通过 Java 访问服务

```
CloudEnvironment cloudEnvironment = new CloudEnvironment();

Collection<RedisServiceInfo> serviceInfos =
    cloudEnvironment.getServiceInfos(RedisServiceInfo.class);

assert serviceInfos.size() > 0 : "there must be at least one bound Redis instance!";

RedisServiceInfo serviceInfo = serviceInfos.iterator().next(); // get the first one

RedisServiceCreator serviceCreator = new RedisServiceCreator();
RedisConnectionFactory cf = serviceCreator.createService(serviceInfo);
```

通过 Java 访问服务

```
CloudEnvironment e = new CloudEnvironment();

Iterable<RdbmsServiceInfo> dsServiceInformation =
    e.getServiceInfos( RdbmsServiceInfo.class) ;

Iterable<RedisServiceInfo> redisServiceInformation =
    e.getServiceInfos( RedisServiceInfo.class) ;

Iterable<RabbitServiceInfo> rabbitServiceInformation =
    e.getServiceInfos( RabbitServiceInfo.class) ;

Iterable<MongoServiceInfo> mongoServiceInformation =
    e.getServiceInfos( MongoServiceInfo.class) ;
```

通过 Spring Java 配置访问服务

```
@Bean public RedisConnectionFactory redis() {

    CloudEnvironment cloudEnvironment = new CloudEnvironment();
    Collection<RedisServiceInfo> serviceInfos =
        cloudEnvironment.getServiceInfos(RedisServiceInfo.class);
    assert serviceInfos.size() > 0 : "there must be at least one bound Redis instance!";
    RedisServiceInfo serviceInfo = serviceInfos.iterator().next(); // get the first one
    RedisServiceCreator serviceCreator = new RedisServiceCreator();
    RedisConnectionFactory cf = serviceCreator.createService(serviceInfo);
    return cf;
}
```

vmware | SPRING & CLOUD FOUNDRY 开发者大会

构建针对 Cloud Foundry 的 Spring 应用程序

明确地通过 XML 绑定至服务

云命名空间简介

- <cloud:> 命名空间在 Spring 应用程序上下文中使用
- 提供 Bean 服务绑定的应用程序级控制
- 推荐用于开发新的云应用程序
- 在以下情况下使用：
  - 具有相同类型的多个服务
  - 具有相同类型的多个 Bean
    - 例如 DataSource、MongoDBFactory
  - 具有自定义 Bean 配置
    - 例如 DataSource 池大小、连接属性

<cloud:service-scan>

- 扫描绑定至应用程序的所有服务并为每个服务创建应用程序类型的
  - 与自动重新配置相同的 Bean 类型
- 在早期开发阶段十分有用

```
<beans ...
    xmlns:cloud="http://schema.cloudfoundry.org/spring"
     xsi:schemaLocation="http://schema.cloudfoundry.org/spring
     http://schema.cloudfoundry.org/spring/cloudfoundry-spring-0.8.xsd
     ...">

       <cloud:service-scan/>

</beans>
```

<cloud:service-scan> 自动关联依赖项

- 创建可以自动关联为依赖项的 Bean
- 如果有多个相同类型的服务绑定到应用程序，则使用带有服务名称的 @Qualifier

```
@Autowired(required=false)
private ConnectionFactory rabbitConnectionFactory;

@Autowired
private RedisConnectionFactory redisConnectionFactory;

@Autowired
@Qualifier("test_mysql_database")
private DataSource mysqlDataSource;

@Autowired(required=false)
@Qualifier("test_postgres_database")
private DataSource postgresDataSource;
```

<cloud:data-source>

- 配置 DataSource Bean
  - Commons DBCP 或 Tomcat DataSource
- 基本属性：
  - ID：服务名称默认值
  - 服务名称：仅在将多个关系数据库服务绑定到应用程序时需要

```
<cloud:data-source id="dataSource" service-name="mySQLSvc">
    <cloud:pool pool-size="1-5"/>
    <cloud:connection properties="charset=utf-8"/>
</cloud:data-source>

<bean class="org.sf.orm.jpa.LocalContainerEntityManagerFactoryBean"
id="entityManagerFactory">
     <property name="dataSource" ref="dataSource"/>
</bean>
```

<cloud:properties>

- 公开有关可通过 Spring 的属性占位符支持使用的服务的基本信息
- 基本属性：
  - ID：属性 Bean 的名称
- 属性在部署 Spring 3.1 应用程序时自动可用

```
<cloud:properties id="cloudProperties"/>
<context:property-placeholder properties-ref="cloudProperties"/>
@Autowired private Environment environment;

@Bean
public ComboPooledDataSource dataSource() throws Exception {
  String user = this.environment.getProperty
        ("cloud.services.mysql.connection.username");
  ComboPooledDataSource cpds  = new  ComboPooledDataSource();
  cpds.setUser(user);
  return cpds;
}
```

vmware | SPRING & CLOUD FOUNDRY 开发者大会

构建针对 Cloud Foundry 的 Spring 应用程序

Spring 环境抽象

Spring 3.1 环境抽象

- 特定环境的 Bean 定义（配置文件）
  - 例如开发、测试、生产
  - 可能不同的部署环境
  - 通过名称激活配置文件
    - **spring.profiles.active** 系统属性
    - 部署单元外的其他方式
    - 如果未指定配置文件，则将激活“默认”配置文件
- 自定义占位符解析
  - 取决于实际环境
  - 有序的属性源

隔离 Cloud Foundry 配置

- 通过配置文件在本地、测试和 Cloud Foundry 部署间切换
- 除非已激活其他配置文件，否则 Spring 自动激活“默认”配置文件
- “云”配置文件自动在 Cloud Foundry 上激活
  - 云命名空间的使用应在云配置文件块内进行

隔离 Cloud Foundry 配置

```
@Configuration
@Profile("default")
public class LocalConfiguration {
   @Bean
   public DataSource dataSource() {
      BasicDataSource bds = new BasicDataSource();
      // ...
      return bds;
   }
}
```

```
@Configuration
@Profile("cloud")
public class CloudConfiguration {
   private CloudEnvironment cloudEnvironment = new CloudEnvironment();

   @Bean
   public DataSource dataSource() {
      Collection<RdbmsServiceInfo> rdbmsServiceInfo = cloudEnvironment.getServiceInfos(RdbmsServiceInfo.class);
      RdbmsServiceCreator rdbmsServiceCreator = new RdbmsServiceCreator();
      return rdbmsServiceCreator.createService(rdbmsServiceInfo.iterator().next());
   }
}
```

隔离 Cloud Foundry 配置

```
<bean class="org.sf.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <property name="dataSource" ref="dataSource"/>
</bean>

<beans profile="cloud">
      <cloud:data-source id="dataSource" />
</beans>

<beans profile="default">
      <bean class="org.a.commons.dbcp.BasicDataSource" id="dataSource">
            <property name="url" value="jdbc:mysql://localhost/my_db" />
      </bean>
</beans>
```

云属性

- Cloud Foundry 使用环境抽象以自动将属性公开至 Spring 3.1 应用程序
  - 应用程序的基本信息，例如名称和云提供商
  - 绑定服务的详细连接信息
    - cloud.services.{service-name}.connection.{property}
    - 根据服务类型创建的服务名称别名
      - 例如“**cloud.services.mysql.connection.{property}**”
      - 仅在存在此类型绑定的单个服务时

云属性示例

- 使用服务属性创建您自己的连接工厂
  - 例如 c3p0 连接池

```
import com.mchange.v2.c3p0.* ;

@Autowired
private Environment  e;

@Bean
public ComboPooledDataSource cpds (){
 ComboPooledDataSource cpds = new ComboPooledDataSource ();

 String host=e.getProperty("cloud.services.mysql.connection.host"),
       port=e.getProperty("cloud.services.mysql.connection.port"),
       name=e.getProperty("cloud.services.mysql.connection.name"),
       pw=e.getProperty("cloud.services.mysql.connection.password");

 cpds.set...( host ... );
 return cpds;
}
```

vmware | SPRING & CLOUD FOUNDRY 开发者大会

构建针对 Cloud Foundry 的 Spring 应用程序

基于 Cloud Foundry 使用 Spring 和 NoSQL

关系数据库十分强大...

- SQL
  - 高级别
  - 排序
  - 聚合
- ACID 语义
- 良好支持
  - JDBC
  - Hibernate/JPA
  - Spring
- 易于理解

... 但也存在限制

- 对象/关系阻抗不匹配
- 富域模型与关系架构间的映射较复杂
- 难于处理半结构化数据，例如不断变化的属性
- 架构更改
- 无法/极难扩展
- 某些情形下的性能极低

解决方案：加大投入

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

- 雇佣更多的开发运营人员
- 使用应用程序级分区
- 构建您自己的中间件

或者

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

解决方案：使用 NoSQL

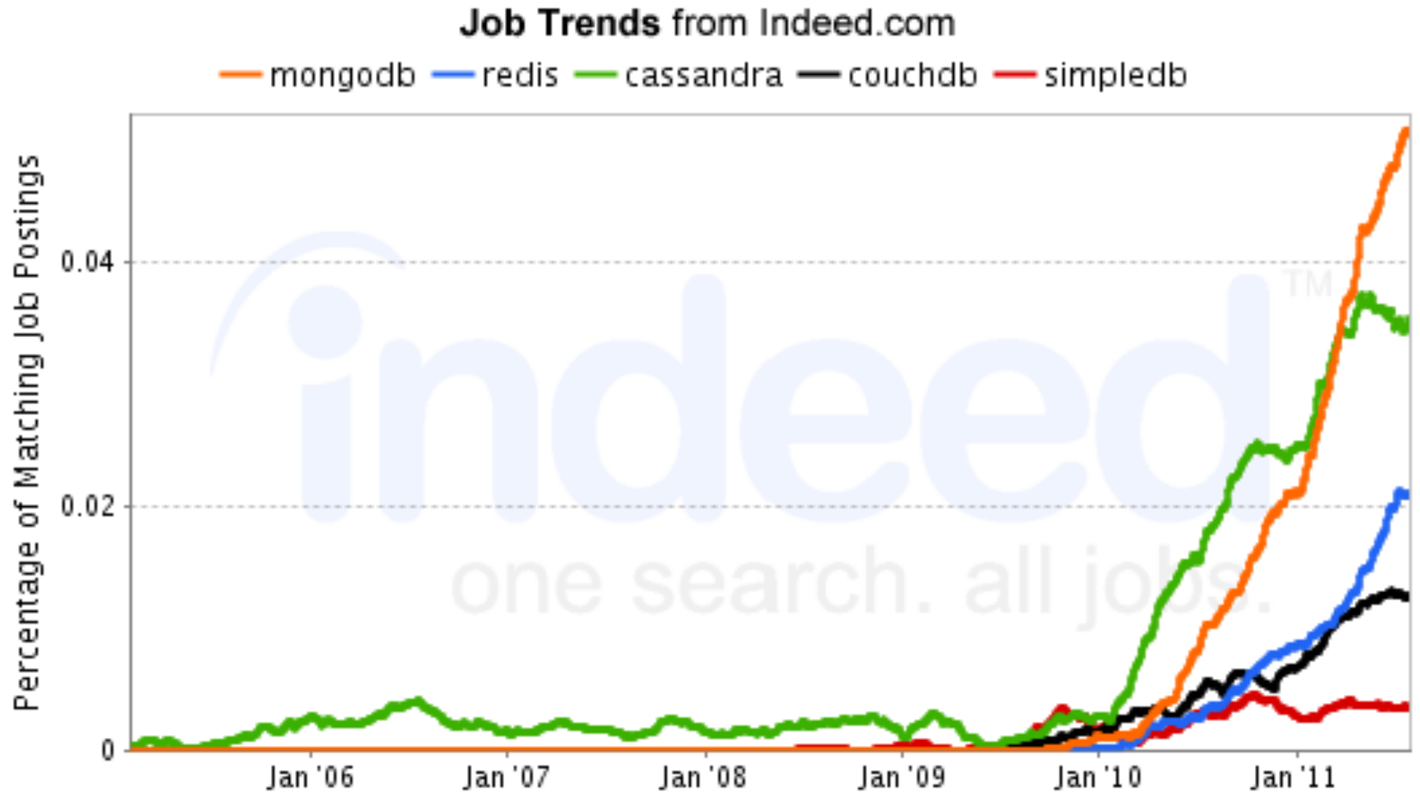
优点

- 更高的性能
- 更高的可扩展性
- 更丰富的数据模型

缺点

- 有限的事务
- 松弛的一致性
- 不受约束的数据

日渐流行...

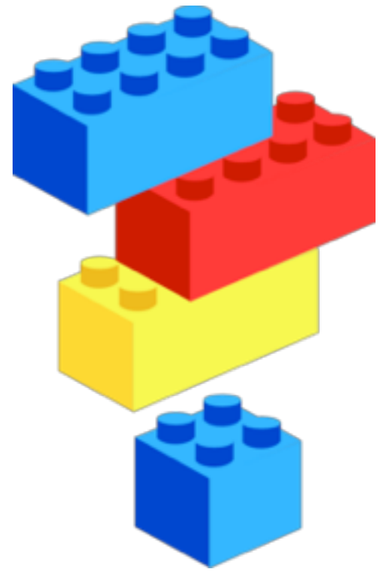

Job Trends from Indeed.com
mongodb
redis
cassandra
couchdb
simpledb
Percentage of Matching Job Postings
0.04
0.02
0
Jan '06
Jan '07
Jan '08
Jan '09
Jan '10
Jan '11
indeed
one search. all jobs.

http://www.springsource.org/spring-data

- **Spring Data 键值**
- **Spring Data 文档**
- **Spring Data 图形**
- Spring Data 列
- Spring Data Blob
- Spring Data JPA 资源库/JDBC 扩展
- Spring Gemfire/Spring Hadoop ...
- Grails i**N**c**O**n**S**e**Q**uentia**L**

Spring Data 构建基础模块

- 低级别数据访问 API
  - ✓MongoTemplate、RedisTemplate ...
- 对象映射（Java 和 GORM）
- 跨存储持久性编程模型
- 通用资源库支持
- Roo 和 Grails 中的高效性支持

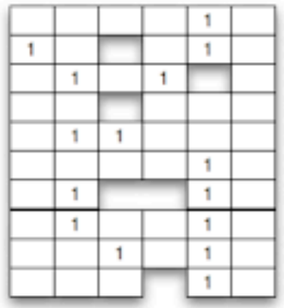


具有 Redis 的 NoSQL

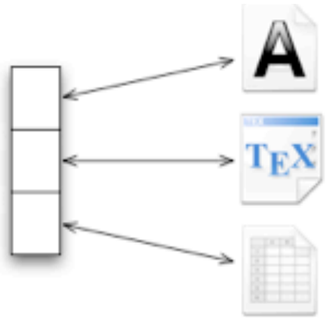
NoSQL 提供几种数据存储类别

键值

列



文档



图形

Redis,
Riak

Spring Data Redis

- **使用 Redis**
  - 非常快
  - “数据结构服务器”（映射、列表、集、队列等）
- **RedisTemplate 将冗长的样板代码减少为单个衬套**
- **CacheManager 实施**
  - 使用 Spring 3.1 的 CacheManager 实施
- **RedisMessageListenerContainer**
  - 提供与 JMS 和 AMQP 相同的队列/发布-订阅功能

配置 Redis 支持

```
@Bean
 public RedisConnectionFactory redisConnectionFactory() {
    return new JedisConnectionFactory();
 }

 @Bean
 public RedisTemplate<String, Object> redisTemplate()
                throws Exception {
    RedisTemplate<String, Object> ro = new RedisTemplate<String, Object>();
    ro.setConnectionFactory(redisConnectionFactory());
    return ro;
 }
```

使用 Redis 处理持久性责任...

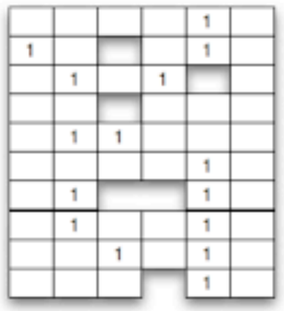
```
@Inject RedisTemplate redisTemplate;

@Override
public Customer getCustomerById(long id) {
   String ln = (String)redisTemplate.opsForValue().get(lastNameKey(id)) ;
   String fn = (String)redisTemplate.opsForValue().get(firstNameKey(id));
   return new Customer(id, fn, ln);
}

private void setCustomerValues(long lid, String fn, String ln) {
   this.redisTemplate.opsForValue().set(lastNameKey(lid), ln);
   this.redisTemplate.opsForValue().set(firstNameKey(lid), fn);
}

@Override
public Customer updateCustomer(long id, String fn, String ln) {
   setCustomerValues(id, fn, ln);
   return getCustomerById(id);
}
```

具有 MongoDB 的 NoSQL

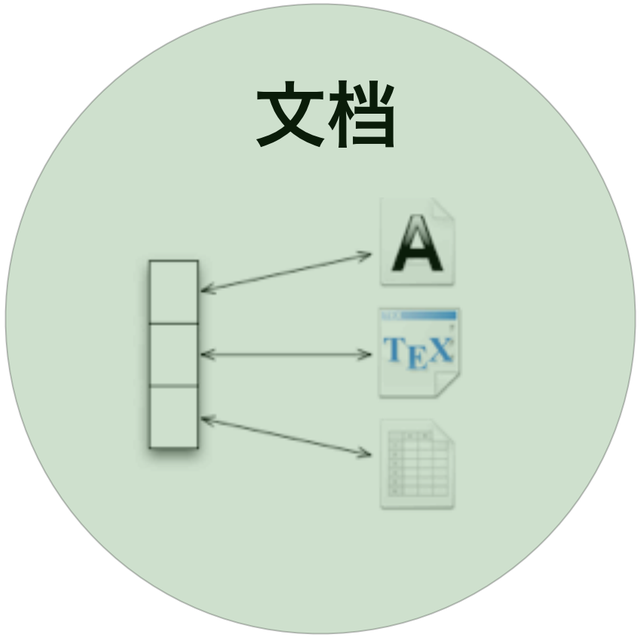
NoSQL 提供几种数据存储类别

键值

列



文档



图形

MongoDB

\

Spring Data MongoDB

- **使用 MongoDB**
- **提供显而易见的 API 集成：**
  - MongoTemplate
  - 以 Mongo 为后盾的 MessageStore（Spring 集成）
- **同样具有高级 API 集成：**
  - 跨存储持久性
  - MongoRepository（基于 Spring Data JPA 构建）

简单的域类

```
public class Person {

  private String id;
  private String name;
  private int age;

  public Person(String name, int age) {
    this.name = name;
    this.age = age;
  }

  public String getId() {
    return id;
  }
  public String getName() {
    return name;
  }
  public int getAge() {
    return age;
  }
```

Mongo 模板

直接使用 Mongo 模板：

```
public class MongoApp {

  private static final Log log = LogFactory.getLog(MongoApp.class);

  public static void main(String[] args) throws Exception {

    MongoOperations mongoOps = new MongoTemplate(new Mongo(), "database");

    mongoOps.insert(new Person("Joe", 34));

    log.info(mongoOps.findOne(new Query(where("name").is("Joe")), Person.class));

    mongoOps.dropCollection("person");
  }
}
```

Mongo 模板

直接使用 Mongo 模板：

```
public class MongoApp {

  private static final Log log = LogFactory.getLog(MongoApp.class);

  public static void main(String[] args) throws Exception {

    MongoOperations mongoOps = new MongoTemplate(new Mongo(), "database");

    mongoOps.insert(new Person("Joe", 34));

    log.info(mongoOps.findOne(new Query(where("name").is("Joe")), Person.class));

    mongoOps.dropCollection("person");
  }
}
```

插入“Person”集合

Mongo 模板

直接使用 Mongo 模板：

```
public class MongoApp {

  private static final Log log = LogFactory.getLog(MongoApp.class);

  public static void main(String[] args) throws Exception {

    MongoOpera                                          go(), "database");

    mongoOps.i

    log.info(mongoOps.findOne(new Query(where("name").is("Joe")), Person.class));

    mongoOps.dropCollection("person");
  }
}
```

findOne在 db.collection: database.Person 中
使用 query: { "name" : "Joe"}

Mongo 模板

直接使用 Mongo 模板：

```
public class MongoApp {

  private static final Log log = LogFactory.getLog(MongoApp.class);

  public static void main(String[] args) throws Exception {

    MongoOperations mongoOps = new MongoTemplate(new Mongo(), "database");

    mongoOp

    log.inf                                                    s("Joe")), Person.class));

    mongoOps.dropCollection("person");
  }
}
```

删除集合 [database.person]

通用资源库

在特定类别资源库上进行通用 CRUD 操作的界面

```
public interface CrudRepository<T, ID extends Serializable>
    extends Repository<T, ID> {

    T save(T entity);

    T findOne(ID primaryKey);

    Iterable<T> findAll();

    Long count();

    void delete(T entity);

    boolean exists(ID primaryKey);

    // … more functionality omitted.
}
```

切换并排序资源库

切换并排序资源库：

扩展“CrudRepository”

```
public interface PagingAndSortingRepository<T, ID extends Serializable> extends CrudRepository<T, ID> {

    Iterable<T> findAll(Sort sort);

    Page<T> findAll(Pageable pageable);
}
```

用法：

切换并排序资源库

切换并排序资源库：

扩展“CrudRepository”

```
public interface PagingAndSortingRepository<T, ID extends Serializable> extends CrudRepository<T, ID> {

    Iterable<T> findAll(Sort sort);

    Page<T> findAll(Pageable pageable);
}
```

用法：

```
PagingAndSortingRepository<User, Long> repository = // … get access to a bean
Page<User> users = repository.findAll(new PageRequest(1, 20);
```

自定义资源库

自定义资源库：

```
public interface PersonRepository extends PagingAndSortingRepository<Person, String> {

    List<Person> findByLastname(String lastname);

    Page<Person> findByFirstname(String firstname, Pageable pageable);

    Person findByShippingAddresses(Address address);

}
```

关键词：

自定义资源库

自定义资源库：

```
public interface PersonRepository extends PagingAndSortingRepository<Person, String> {

    List<Person> findByLastname(String lastname);

    Page<Person> findByFirstname(String firstname, Pageable pageable);

    Person findByShippingAddresses(Address address);

}
```

关键词：

关键词	示例	逻辑结果
GreaterThan	findByAgeGreaterThan(int age)	{"age" : {"$gt" : age}}
LessThan	findByAgeLessThan(int age)	{"age" : {"$lt" : age}}
Between	findByAgeBetween(int from, int to)	{"age" : {"$gt" : from, "$lt" : to}}
NotNull	findByFirstnameNotNull()	{”firstname" : {"$ne" : null}}
Null	findByFirstnameNull()	{”firstname" : null}
Like	findByFirstnameLike(String name)	"firstname" : firstname} (regex)

JPA 和 MongoDB 可在跨存储持久性中使用

具有“SurveyInfo”文档的 JPA“Customer”

```
@Entity
public class Customer {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  private String firstName;

  private String lastName;

  @RelatedDocument
  private SurveyInfo surveyInfo;

  // getters and setters omitted

}
```

使用跨存储功能

保存具有 SurveryInfo 的 Customer

```
Customer customer = new Customer();
customer.setFirstName("Sven");
customer.setLastName("Olafsen");
SurveyInfo surveyInfo = new SurveyInfo()
  .addQuestionAndAnswer("age", "22")
  .addQuestionAndAnswer("married", "Yes")
  .addQuestionAndAnswer("citizenship", "Norwegian");
customer.setSurveyInfo(surveyInfo);
customerRepository.save(customer);
```

Mongo 文档：

使用跨存储功能

保存具有 SurveryInfo 的 Customer

```
Customer customer = new Customer();
customer.setFirstName("Sven");
customer.setLastName("Olafsen");
SurveyInfo surveyInfo = new SurveyInfo()
  .addQuestionAndAnswer("age", "22")
  .addQuestionAndAnswer("married", "Yes")
  .addQuestionAndAnswer("citizenship", "Norwegian");
customer.setSurveyInfo(surveyInfo);
customerRepository.save(customer);
```

Mongo 文档：

```
{ "_id" : ObjectId( "4d9e8b6e3c55287f87d4b79e" ),
  "_entity_id" : 1,
  "_entity_class" : "org.springframework.data.mongodb.examples.custsvc.domain.Customer",
  "_entity_field_name" : "surveyInfo",
  "questionsAndAnswers" : { "married" : "Yes",
    "age" : "22",
    "citizenship" : "Norwegian" },
  "_entity_field_class" : "org.springframework.data.mongodb.examples.custsvc.domain.SurveyInfo" }
```

使用跨存储功能

保存具有 SurveryInfo 的 Customer

创建 Customer

```
Customer customer = new Customer();
customer.setFirstName("Sven");
customer.setLastName("Olafsen");
SurveyInfo surveyInfo = new SurveyInfo()
  .addQuestionAndAnswer("age", "22")
  .addQuestionAndAnswer("married", "Yes")
  .addQuestionAndAnswer("citizenship", "Norwegian");
customer.setSurveyInfo(surveyInfo);
customerRepository.save(customer);
```

Mongo 文档：

```
{ "_id" : ObjectId( "4d9e8b6e3c55287f87d4b79e" ),
  "_entity_id" : 1,
  "_entity_class" : "org.springframework.data.mongodb.examples.custsvc.domain.Customer",
  "_entity_field_name" : "surveyInfo",
  "questionsAndAnswers" : { "married" : "Yes",
    "age" : "22",
    "citizenship" : "Norwegian" },
  "_entity_field_class" : "org.springframework.data.mongodb.examples.custsvc.domain.SurveyInfo" }
```

使用跨存储功能

保存具有 SurveryInfo 的 Customer

```
Customer customer = new
customer.setFirstName("S
customer.setLastName("Ol
SurveyInfo surveyInfo = new SurveyInfo()
  .addQuestionAndAnswer("age", "22")
  .addQuestionAndAnswer("married", "Yes")
  .addQuestionAndAnswer("citizenship", "Norwegian");
customer.setSurveyInfo(surveyInfo);
customerRepository.save(customer);
```

创建 SurveyInfo

Mongo 文档：

```
{ "_id" : ObjectId( "4d9e8b6e3c55287f87d4b79e" ),
  "_entity_id" : 1,
  "_entity_class" : "org.springframework.data.mongodb.examples.custsvc.domain.Customer",
  "_entity_field_name" : "surveyInfo",
  "questionsAndAnswers" : { "married" : "Yes",
    "age" : "22",
    "citizenship" : "Norwegian" },
  "_entity_field_class" : "org.springframework.data.mongodb.examples.custsvc.domain.SurveyInfo" }
```

使用跨存储功能

保存具有 SurveryInfo 的 Customer

```
Customer customer = new Customer();
customer.setFirstName("Sven");
customer.setLastName("Olafsen");
SurveyIn                                    ()
   .addQue
   .addQue                                  es")
   .addQuestionA      swer("citizenship", "Norwegian");
customer.setSurveyInfo(surveyInfo);
customerRepository.save(customer);
```

将 Survey 分配给 Customer

Mongo 文档：

```
{ "_id" : ObjectId( "4d9e8b6e3c55287f87d4b79e" ),
  "_entity_id" : 1,
  "_entity_class" : "org.springframework.data.mongodb.examples.custsvc.domain.Customer",
  "_entity_field_name" : "surveyInfo",
  "questionsAndAnswers" : { "married" : "Yes",
    "age" : "22",
    "citizenship" : "Norwegian" },
  "_entity_field_class" : "org.springframework.data.mongodb.examples.custsvc.domain.SurveyInfo" }
```

使用跨存储功能

保存具有 SurveryInfo 的 Customer

```
Customer customer = new Customer();
customer.setFirstName("Sven");
customer.setLastName("Olafsen");
SurveyInfo surveyInfo = new SurveyInfo()
  .addQuestion
  .addQuestion                                    s")
  .addQuestion                                     "Norwegian");
customer.setSu
customerRepository.save(customer);
```

保存

Mongo 文档：

```
{ "_id" : ObjectId( "4d9e8b6e3c55287f87d4b79e" ),
  "_entity_id" : 1,
  "_entity_class" : "org.springframework.data.mongodb.examples.custsvc.domain.Customer",
  "_entity_field_name" : "surveyInfo",
  "questionsAndAnswers" : { "married" : "Yes",
    "age" : "22",
    "citizenship" : "Norwegian" },
  "_entity_field_class" : "org.springframework.data.mongodb.examples.custsvc.domain.SurveyInfo" }
```

vmware | SPRING & CLOUD FOUNDRY 开发者大会

构建针对 Cloud Foundry 的 Spring 应用程序

基于Cloud Foundry 使用 RabbitMQ传递消息

jlong — bash — 100×21

```
Password: *******
Successfully logged into [http://api.cloudfoundry.com]

jlongmbp17:~ jlong$ vmc services

============== System Services ==============

+------------+---------+-----------------------------------------+
| Service    | Version | Description                             |
+------------+---------+-----------------------------------------+
| atmos      | 1.4.1   | Atmos object store                      |
| mongodb    | 1.8     | MongoDB NoSQL store                     |
| mysql      | 5.1     | MySQL database service                  |
| postgresql | 9.0     | PostgreSQL database service             |
| rabbitmq   | 2.4     | RabbitMQ messaging service              |
| redis      | 2.2     | Redis key-value store service           |
+------------+---------+-----------------------------------------+

============ Provisioned Services ============

+---------------------+------------+
```

原始的“云规模”消息

消息使用情形

解耦

购物车向云控制器商家发送请求

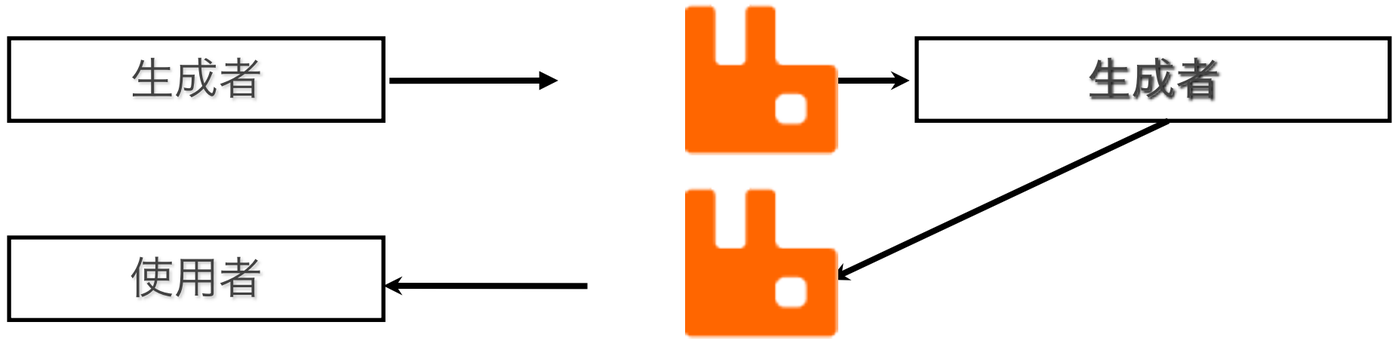
消息使用情形

双向解耦

例如：远程过程调用

消息使用情形

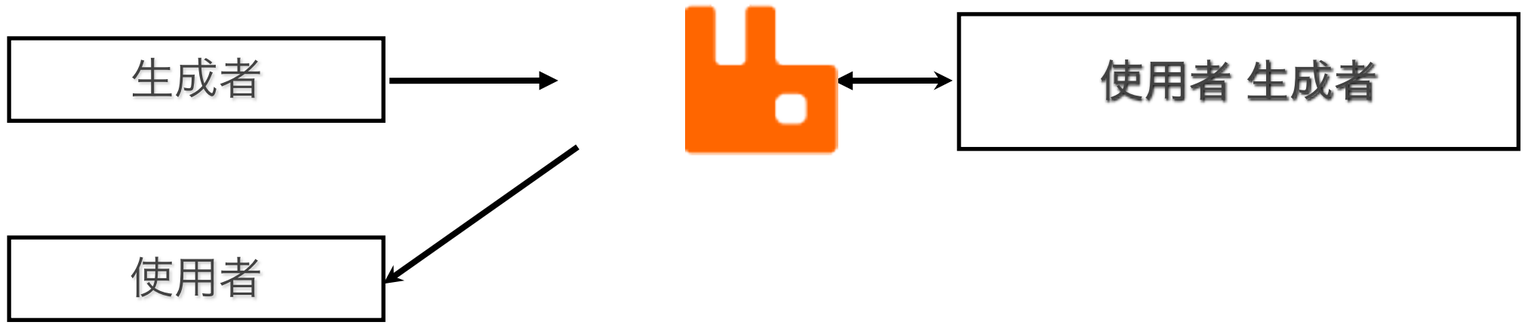
双向解耦



例如：下订单和等待确认

消息使用情形

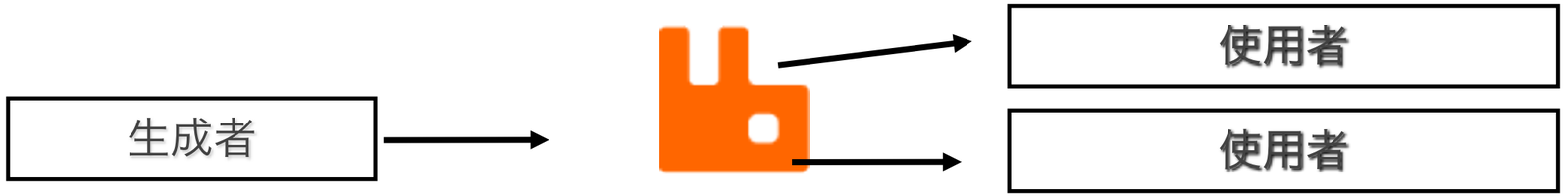
双向解耦



例如：下订单和等待确认

消息使用情形

工作分发和解耦



分发可以重复或循环：

- 复制信息（记录、审核等）
- 循环扩展和负载平衡

AMQP

- 发送和接收 AMQP 消息

```
@Component
public class MessageSender {

  @Autowired
  private AmqpTemplate amqpTemplate;

  public void send(String message) {
    this.amqpTemplate.convertAndSend(
          "myExchange", "some.routing.key", message);
  }

}
```

```
@Component
public class MessageReceiver {

  @Autowired
  private RabbitTemplate rabbitTemplate;

  public void read() throws Exception {
     String value = rabbitTemplate.receiveAndConvert("myQueueName");
  }

}
```

@starbuxman | josh.long@springsource.com

http://slideshare.net/joshlong

问题？