

# O/R Mapping 实践

# 认识 Spring Data JPA

# 对象与关系的范式不匹配

	Object	RDBMS
粒度	类	表
继承	有	没有
唯一性	<code>a == b</code> <code>a.equals(b)</code>	主键
关联	引用	外键
数据访问	逐级访问	SQL 数量要少

# Hibernate

- 一款开源的对象关系映射（Object / Relational Mapping）框架
- 将开发者从 95% 的常见数据持久化工作中解放出来
- 屏蔽了底层数据库的各种细节

# Hibernate 发展历程

- 2001年, Gavin King 发布第一个版本
- 2003年, Hibernate 开发团队加入 JBoss
- 2006年, Hibernate 3.2 成为 JPA 实现

# Java Persistence API

**JPA 为对象关系映射提供了一种基于 POJO 的持久化模型** Plain Ordinary Java Object

JPA就是Hi bernate

- 简化数据持久化代码的开发工作
- 为 Java 社区屏蔽不同持久化 API 的差异

2006 年, *JPA 1.0* 作为 *JSR 220* 的一部分正式发布

# Spring Data

Spring Framework中剥离出来的子模块

在保留底层存储特性的同时，提供相对一致的、基于 **Spring** 的编程模型

## 主要模块

- Spring Data Commons
- Spring Data JDBC
- **Spring Data JPA**
- Spring Data Redis
- .....

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.data</groupId>
      <artifactId>spring-data-releasetrain</artifactId>
      <version>Lovelace-SR4</version>
      <scope>import</scope>
      <type>pom</type>
    </dependency>
  </dependencies>
</dependencyManagement>
```

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

# 定义 JPA 实体对象



# 常用 JPA 注解

## 实体

- @Entity、@MappedSuperclass

注明类时是个实体

表明是多个实体的父类

- @Table(name) 实体和对应的表关联起来

## 主键

- @Id
  - @GeneratedValue(strategy, generator)
  - @SequenceGenerator(name, sequenceName)

# 常用 JPA 注解

```
@Entity(name = "Product")
public static class Product {

    @Id
    @GeneratedValue(
        strategy = GenerationType.SEQUENCE,
        generator = "sequence-generator"
    )
    @SequenceGenerator(
        name = "sequence-generator",
        sequenceName = "product_sequence"
    )
    private Long id;

    @Column(name = "product_name")
    private String name;

    //Getters and setters are omitted for brevity

}
```

# 常用 JPA 注解

## 映射

- @Column(name, nullable, length, insertable, updatable)
- @JoinTable(name)、@JoinColumn(name) 表的关联使用

## 关系

- @OneToOne、@OneToMany、@ManyToOne、@ManyToMany
- @OrderBy

# Project Lombok

**Project Lombok** 能够自动嵌入 IDE 和构建工具，提升开发效率

## 常用功能

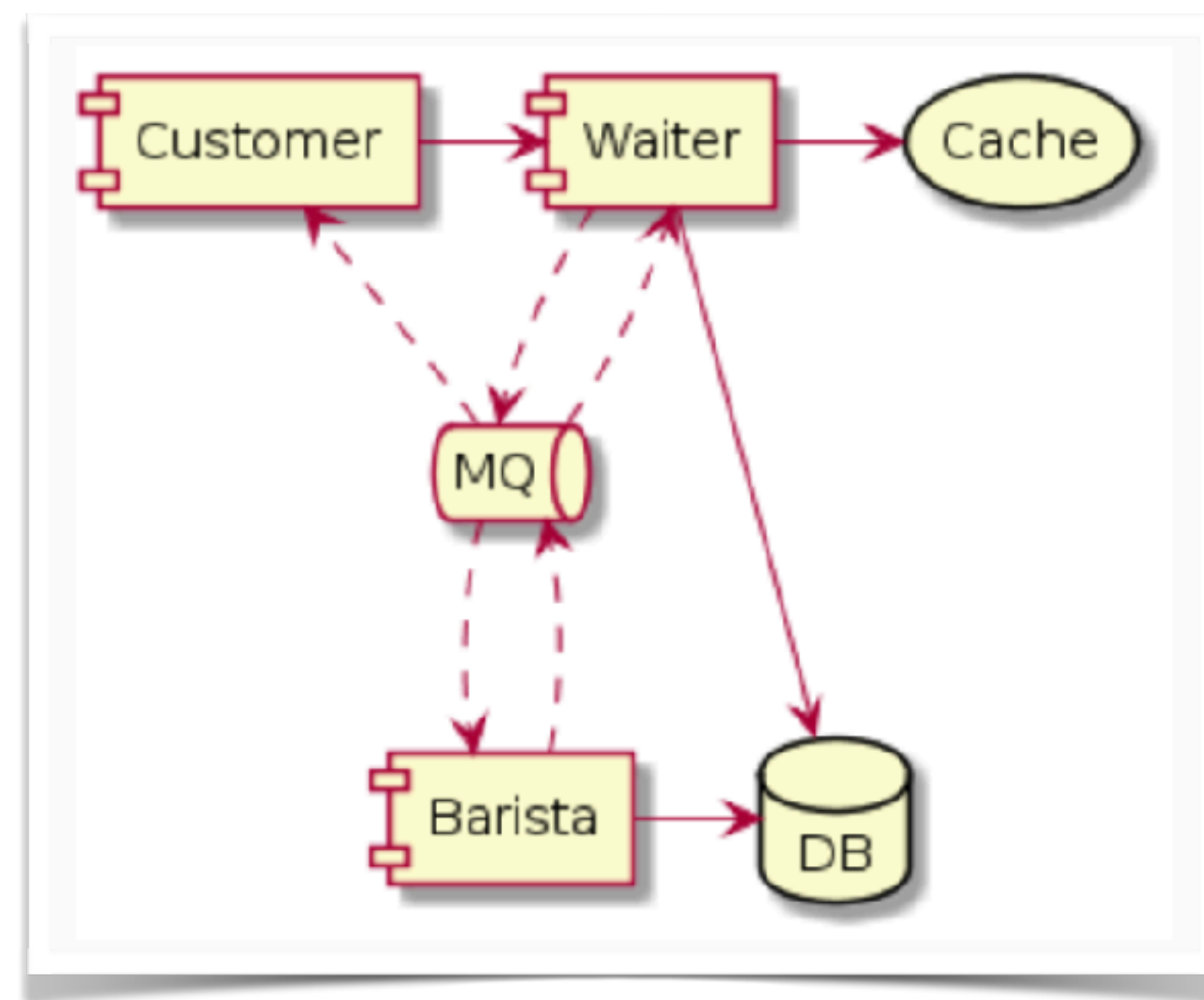
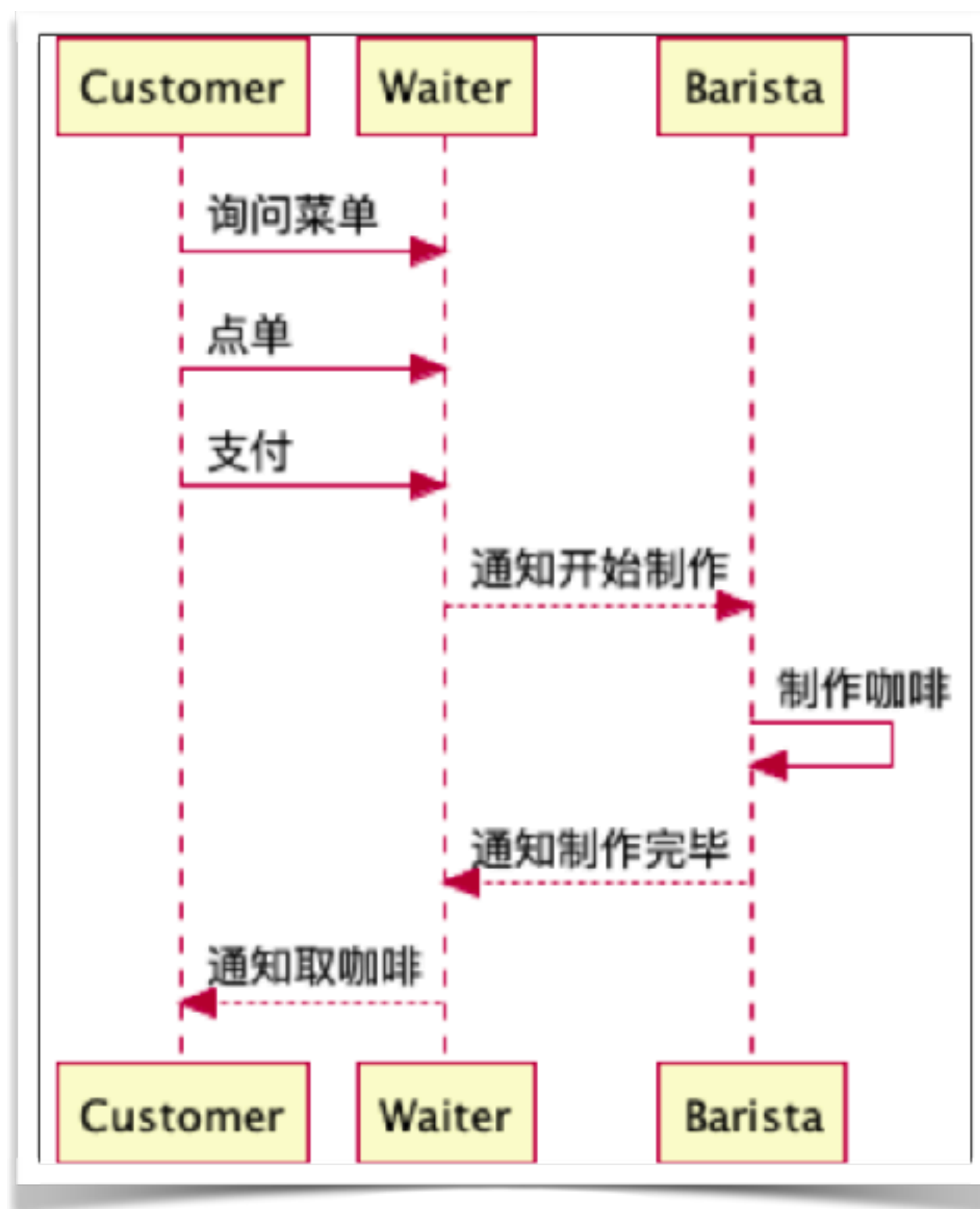
- @Getter / @Setter      自动生成get() set()
- @ToString      生成对象的完整的属性，并且打印出来
- @NoArgsConstructor / @RequiredArgsConstructor / @AllArgsConstructor
- @Data      @Getter / @Setter      @ToString
- @Builder      生成build()方法，帮助构建对象
- @Slf4j / @CommonsLog / @Log4j2      日志相关注解

# 线上咖啡馆实战项目

SpringBucks

# 项目目标

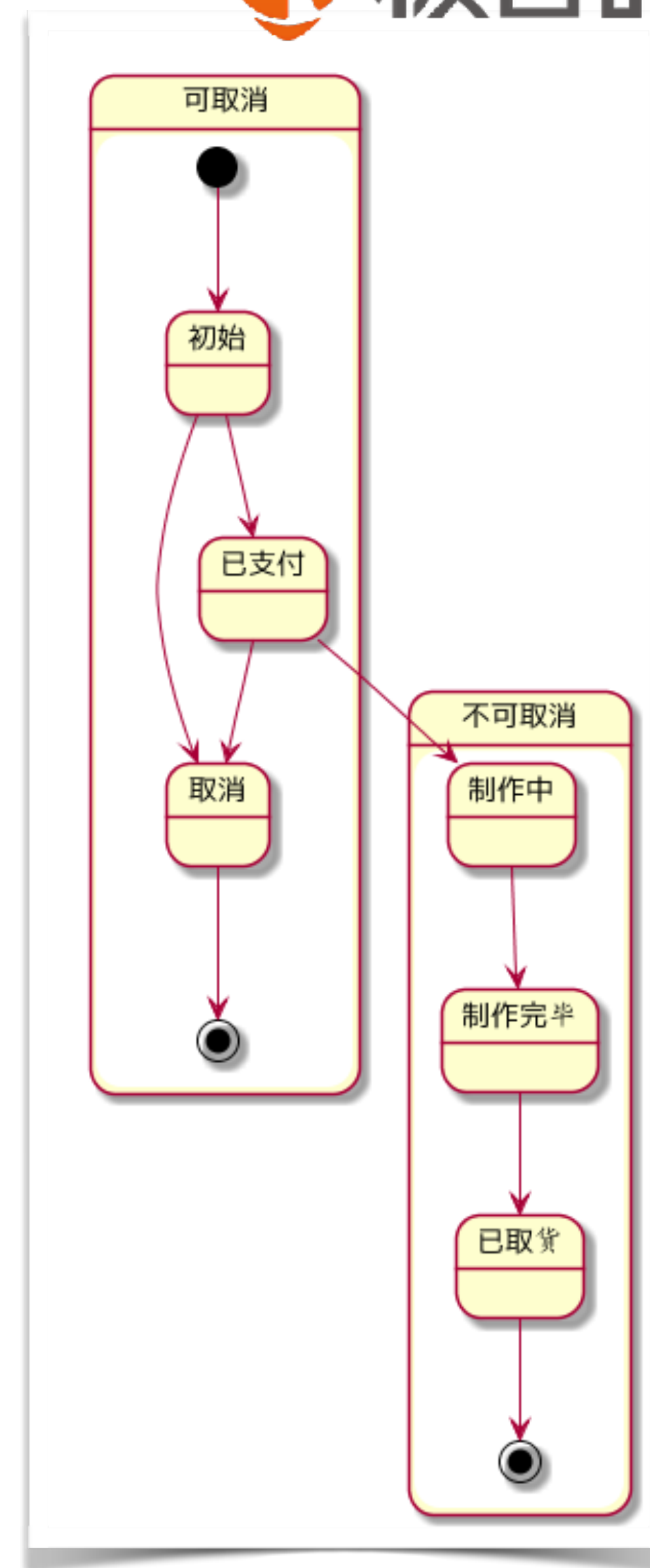
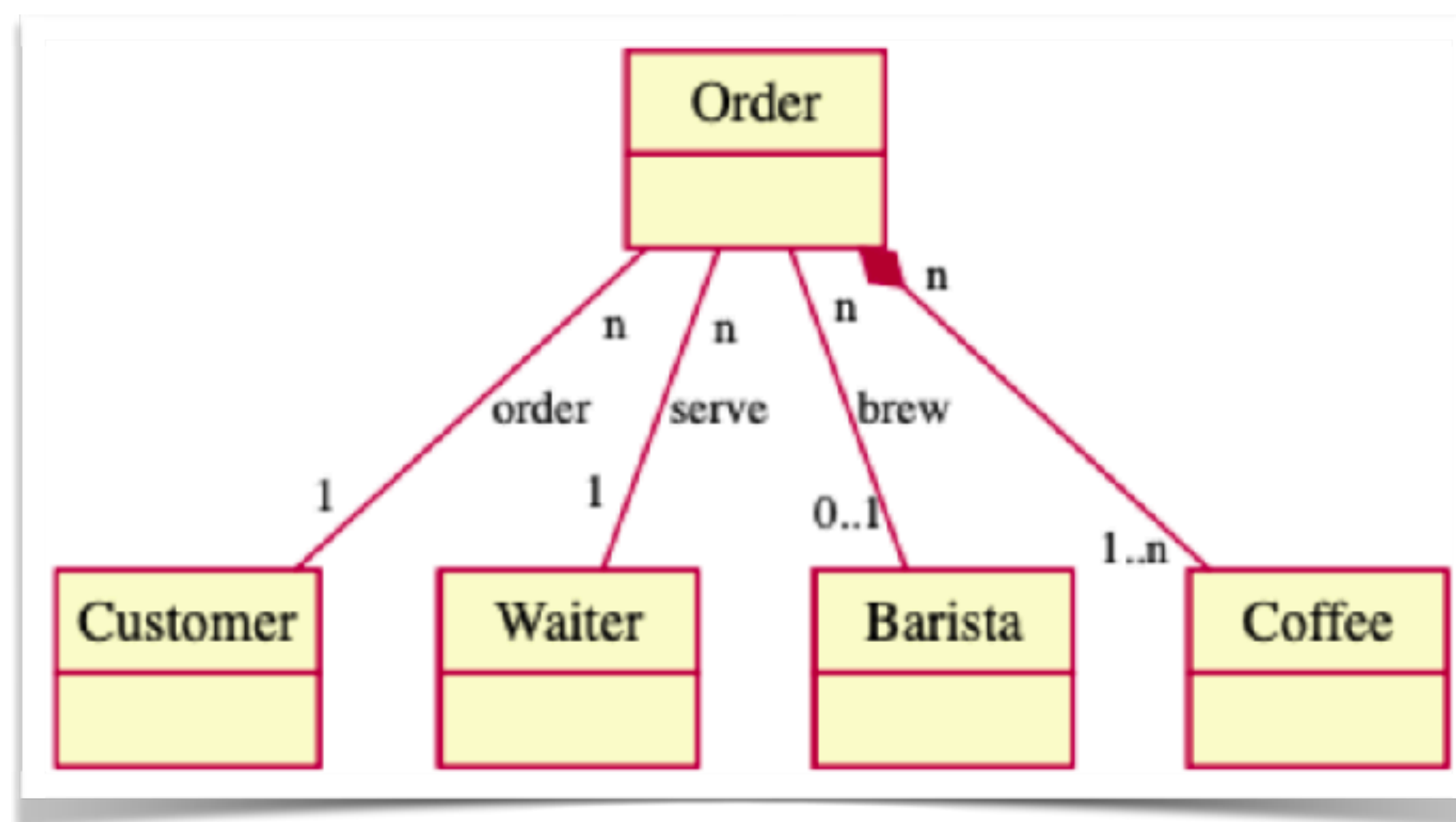
通过一个完整的例子演示 Spring 全家桶各主要成员的用法



# 项目中的对象实体

## 实体

- 咖啡、订单、顾客、服务员、咖啡师





# 实体定义

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

```
<dependency>
  <groupId>org.joda</groupId>
  <artifactId>joda-money</artifactId>
  <version>1.0.1</version>
</dependency>
```

金融领域：一定不会使用浮点数表示金额

joda-money依赖，通过money对象表示金额

usertype.core 作很多映射相关的订制和信息

```
</dependency>
<dependency>
  <groupId>org.jadira.usertype</groupId>
  <artifactId>usertype.core</artifactId>
  <version>6.0.1.GA</version>
</dependency>
```

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
```



# 实体定义

```
@Entity
@Table(name = "T_MENU")
@Builder
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Coffee implements Serializable {
    @Id
    @GeneratedValue
    private Long id;
    private String name;
    @Column
    @Type(type = "org.jadira.usertype.moneyandcurrency.joda.PersistentMoneyAmount",
        parameters = {@org.hibernate.annotations.Parameter(name = "currencyCode", value = "CNY")})
    private Money price;
    @Column(updatable = false)
    @CreationTimestamp
    private Date createTime;
    @UpdateTimestamp
    private Date updateTime;
}
```

Coffee espresso = Coffee.builder().name("espresso")  
.price(Money.of(CurrencyUnit.of("CNY"), 20.0))  
.build();

@Id 注解表明Id属性其实是主键  
主键是自动生成的

@GeneratedValue(strategy = GenerationType.IDENTITY) // 自增的生成策略  
若这里修改成这样的话，就不会生成：  
Hibernate: create sequence hibernate\_sequence start with 1 increment by

表明不可修改

# 实体定义

```
@Entity
@Table(name = "T_ORDER")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class CoffeeOrder implements Serializable {
    @Id
    @GeneratedValue
    private Long id;
    private String customer;
    @ManyToMany
    @JoinTable(name = "T_ORDER_COFFEE")
    private List<Coffee> items;
    @Column(nullable = false)
    private Integer state;
    @Column(updatable = false)
    @CreationTimestamp
    private Date createTime;
    @UpdateTimestamp
    private Date updateTime;
}
```

many to many

@JoinTable 说明了Coffee 和 Order的关系是通过T\_ORDER\_COFFEE这张映射表实现的

**“Talk is cheap, show me the code.”**

*Chapter 3 / jpa-demo* 仔细看这个Demo 和 视频讲解

# 实体定义

```
@MappedSuperclass
@Data
@NoArgsConstructor
@AllArgsConstructor
public class BaseEntity implements Serializable {
    @Id
    @GeneratedValue
    private Long id;
    @Column(updatable = false)
    @CreationTimestamp
    private Date createTime;
    @UpdateTimestamp
    private Date updateTime;
}
```

# 实体定义

```
@Entity
@Table(name = "T_MENU")
@Builder
@Data
@ToString(callSuper = true)    调用BaseEntity里的toString()一起打印，不然只会打印这个类的toString()
@NoArgsConstructor
@AllArgsConstructor
public class Coffee extends BaseEntity implements Serializable {
    private String name;
    @Type(type = "org.jadira.usertype.moneyandcurrency.joda.PersistentMoneyAmount",
        parameters = {@org.hibernate.annotations.Parameter(name = "currencyCode", value = "CNY")})
    private Money price;
}
```



# 实体定义

```
@Entity
@Table(name = "T_ORDER")
@Data
@ToString(callSuper = true)
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class CoffeeOrder extends BaseEntity implements Serializable {
    private String customer;
    @ManyToMany
    @JoinTable(name = "T_ORDER_COFFEE")
    @OrderBy("id")
    private List<Coffee> items;
    @Enumerated
    @Column(nullable = false)
    private OrderState state;
}

public enum OrderState {
    INIT, PAID, BREWING, BREWED, TAKEN, CANCELLED
}
```

**“Talk is cheap, show me the code.”**

*Chapter 3 / jpa-complex-demo*

# 通过 Spring Data JPA 操作数据库



# Repository

使用Spring Data JPA操作数据库，只需要其接口，并不需要接口的实现类

**@EnableJpaRepositories** 开启JPA

## Repository<T, ID> 接口

- CrudRepository<T, ID>
- PagingAndSortingRepository<T, ID>
- JpaRepository<T, ID>

# 定义查询

## 根据方法名定义查询

- find...By... / read...By... / query...By... / get...By...
- count...By...
- ...OrderBy...[Asc / Desc]
- And / Or / IgnoreCase
- Top / First / Distinct

# 分页查询

## 分页查询

- `PagingAndSortingRepository<T, ID>`
- `Pageable / Sort`
- `Slice<T> / Page<T>`

# 保存实体

```
Coffee espresso = Coffee.builder().name("espresso")
    .price(Money.of(CurrencyUnit.of("CNY"), 20.0))
    .build();
coffeeRepository.save(espresso);
log.info("Coffee: {}", espresso);

Coffee latte = Coffee.builder().name("latte")
    .price(Money.of(CurrencyUnit.of("CNY"), 30.0))
    .build();
coffeeRepository.save(latte);
log.info("Coffee: {}", latte);
```

# 保存实体

```
order = CoffeeOrder.builder()
    .customer("Li Lei")
    .items(Arrays.asList(espresso, latte))
    .state(0)
    .build();
orderRepository.save(order);
log.info("Order: {}", order);
```

**“Talk is cheap, show me the code.”**

*Chapter 3 / jpa-demo*

# 查询实体

`@NoRepositoryBean` 说明不需要为BaseRepository创建一个Bean

```
public interface BaseRepository<T, Long> extends PagingAndSortingRepository<T, Long> {  
    List<T> findTop3ByOrderByUpdateTimeDescIdAsc();  
}
```

```
public interface CoffeeOrderRepository extends BaseRepository<CoffeeOrder, Long> {  
    List<CoffeeOrder> findByCustomerOrderId(String customer);  
    List<CoffeeOrder> findByItems_Name(String name);  
}
```

Items\_Name通过\_可以告诉Spring Data，我要访问的是item里的name这个属性，避免混淆(但是也可以不用这个下划线)



# 查询实体

```
coffeeRepository
    .findAll(Sort.by(Sort.Direction.DESC, "id"))
    .forEach(c -> log.info("Loading {}", c));

List<CoffeeOrder> list = orderRepository.findTop3ByOrderByUpdateTimeDescIdAsc(); Coffee Order中找出Top 3个
log.info("findTop3ByOrderByUpdateTimeDescIdAsc: {}", getJoinedOrderId(list));

list = orderRepository.findByCustomerId("Li Lei");
log.info("findByCustomerId: {}", getJoinedOrderId(list));

// 不开启事务会因为没Session而报LazyInitializationException
list.forEach(o -> {
    log.info("Order {}", o.getId());
    o.getItems().forEach(i -> log.info("Item {}", i));
});

list = orderRepository.findByItemsName("latte");
log.info("findByItemsName: {}", getJoinedOrderId(list));
```



**“Talk is cheap, show me the code.”**

*Chapter 3 / jpa-complex-demo*

Repository 是怎么从接口变成 Bean 的

# Repository Bean 是如何创建的

## **JpaRepositoriesRegistrar**

- 激活了 @EnableJpaRepositories
- 返回了 JpaRepositoryConfigExtension

## **RepositoryBeanDefinitionRegistrarSupport.registerBeanDefinitions**

- 注册 Repository Bean (类型是 JpaRepositoryFactoryBean)

## **RepositoryConfigurationExtensionSupport.getRepositoryConfigurations**

- 取得 Repository 配置

## **JpaRepositoryFactory.getTargetRepository**

- 创建了 Repository

# 接口中的方法是如何被解释的

**RepositoryFactorySupport.getRepository 添加了Advice**

- DefaultMethodInvokingMethodInterceptor
- QueryExecutorMethodInterceptor

**AbstractJpaQuery.execute 执行具体的查询**

语法解析在 Part 中

# 通过 MyBatis 操作数据库

# 认识 MyBatis

**MyBatis** (<https://github.com/mybatis/mybatis-3>)

- 一款优秀的持久层框架
- 支持定制化 SQL、存储过程和高级映射
  - 与 Hibernate(JPA)不同的是，需要自己定制 SQL，即自己写 SQL.
  - \* 若 SQL 比较简单，使用 JPA;
  - \* 若 SQL 比较复杂，使用 MyBatis

## 在 Spring 中使用 MyBatis

- MyBatis Spring Adapter (<https://github.com/mybatis/spring>)
- MyBatis Spring-Boot-Starter (<https://github.com/mybatis/spring-boot-starter>)

Dependency

# 简单配置

配置

- `mybatis.mapper-locations = classpath*:mapper/**/*.xml`
- `mybatis.type-aliases-package = 类型别名的包名`
- `mybatis.type-handlers-package = TypeHandler扫描包名`  
指定所有Handler所处的包名
- `mybatis.configuration.map-underscore-to-camel-case = true`

下划线与驼峰映射

# Mapper 的定义与扫描

- @MapperScan 配置扫描位置 `@MapperScan("com.spring.data.mybatisdemo.mapper")`
- @Mapper 定义接口
- 映射的定义——XML 与注解



# Mapper 的定义与扫描

```
@Mapper
public interface CoffeeMapper {
    @Insert("insert into t_coffee (name, price, create_time, update_time)"
            + "values ({name}, #{price}, now(), now())")
    @Options(useGeneratedKeys = true)
    Long save(Coffee coffee);

    @Select("select * from t_coffee where id = #{id}")
    @Results({
        @Result(id = true, column = "id", property = "id"),
        @Result(column = "create_time", property = "createTime"),
        // map-underscore-to-camel-case = true 可以实现一样的效果
        // @Result(column = "update_time", property = "updateTime"),
    })
    // 这其实是将数据库表中属性的表示形式转换为类对象属性的表示方式
    Coffee findById(@Param("id") Long id);
}
```

**“Talk is cheap, show me the code.”**

*Chapter 3 / mybatis-demo*

# 让 MyBatis 更好用的那些工具

## MyBatis Generator

- \* pom.xml dependency
- \* shema.sql
- \* application.properties
- \* resources/mapper 目录
- \* generatorConfig.xml

注意，生成的和手写的最好分开放置，这样的话，就可以分开覆盖，不至于自动生成代码时覆盖掉手写的配置。

# 认识 MyBatis Generator

**MyBatis Generator** (<http://www.mybatis.org/generator/>)

- MyBatis 代码生成器
- 根据数据库表生成相关代码
  - POJO
  - Mapper 接口
  - SQL Map XML

# 运行 MyBatis Generator

## 命令行

- `java -jar mybatis-generator-core-x.x.x.jar -configfile generatorConfig.xml`

## Maven Plugin (mybatis-generator-maven-plugin)

- `mvn mybatis-generator:generate`
- `${basedir}/src/main/resources/generatorConfig.xml`

## Eclipse Plugin

## Java 程序

## Ant Task

# 配置 MyBatis Generator

## **generatorConfiguration**

generatorConfig.xml 中所有配置项都是有顺序的，必须按顺序配置

在resources目录下必须有mapper目录

## **context**

\* plugin

- jdbcConnection
- javaModelGenerator
- sqlMapGenerator
- javaClientGenerator (ANNOTATEDMAPPER / XMLMAPPER / MIXEDMAPPER)
- table

# 生成时可以使用插件

内置插件都在 `org.mybatis.generator.plugins` 包中

- `FluentBuilderMethodsPlugin`
- `ToStringPlugin`
- `SerializablePlugin`
- `RowBoundsPlugin`
- .....

# 使用生成的对象

- 简单操作，直接使用生成的 xxxMapper 的方法
- 复杂查询，使用生成的 xxxExample 对象



# 使用生成的对象

```
Coffee latte = new Coffee()
    .withName("latte")
    .withPrice(Money.of(CurrencyUnit.of("CNY"), 30.0))
    .withCreateTime(new Date())
    .withUpdateTime(new Date());
coffeeMapper.insert(latte);

Coffee s = coffeeMapper.selectByPrimaryKey(1L);
log.info("Coffee {}", s);

CoffeeExample example = new CoffeeExample();
example.createCriteria().andNameEqualTo("latte");
List<Coffee> list = coffeeMapper.selectByExample(example);
list.forEach(e -> log.info("selectByExample: {}", e));
```

**“Talk is cheap, show me the code.”**

*Chapter 3 / mybatis-generator-demo*

# 让 MyBatis 更好用的那些工具

MyBatis PageHelper

# 认识 MyBatis PageHelper

**MyBatis PageHelper** (<https://pagehelper.github.io>)

- 支持多种数据库
- 支持多种分页方式
- SpringBoot 支持 (<https://github.com/pagehelper/pagehelper-spring-boot> )
  - pagehelper-spring-boot-starter

**“Talk is cheap, show me the code.”**

*Chapter 3 / mybatis-pagehelper-demo*

# SpringBucks 进度小结

## 本章小结

- 简单了解了 Java Persistence API 的背景
- 学习了 JPA 的常用注解
- 学习了 Lombok 的用法
- 学习了 Spring Data JPA 的基本用法
- 学习了 MyBatis 及其相关工具的基本用法

# SpringBucks 进度小结

- 概述性地了解了 SpringBucks 的需求
- 通过 JPA 定义并实现了 SpringBucks 中的一些实体
- 通过 Spring Data JPA 定义了一些简单的 Repository



1. Dependency
2. application.properties
3. schema.sql / data.sql
4. model
  - Entity
5. Repository
  - extends JpaRepository<Entity, id>
6. Service

“Talk is cheap, show me the code.”

JpaRepository -> PagingAndSortingRepository -> CrudRepository -> Repository  
QueryByExampleExecutor

*Chapter 3 / springbucks*