# 第一章：初识Spring Data JPA

## 1.ORM框架对比

### mybatis

MyBatis本是Apache 的一个开源项目iBatis, 2010年这个项目由Apache Software Foundation迁移到了Google Code,并且改名为MyBatis。MyBatis着力于POJO与SQL之间的映射关系，可以进行更为细致的SQL，使用起来十分灵活，上手简单， 容易掌握，所以深受开发者的喜欢，目前市场占有率最高，比较适合互联应用公司的API场景。

### hibernate

Hibernate是一个开放源代码的对象关系映射框架，对JDBC进行了非常轻量级的对象封装，使得Java程序员可以随心所欲就地使用对象编程思维来提纵数据库，并且对象有自己的生命周期，着力对象与对象之间的关系，有自己的HQL查询语言，所以数据库移植性很好。Hibemate是完备的ORM框架，是符合JPA规范的。Hibemate有自己的缓存机制。从上手的角度来说比较难，比较适合企业级的应用系统开发。

### Spring Data JPA

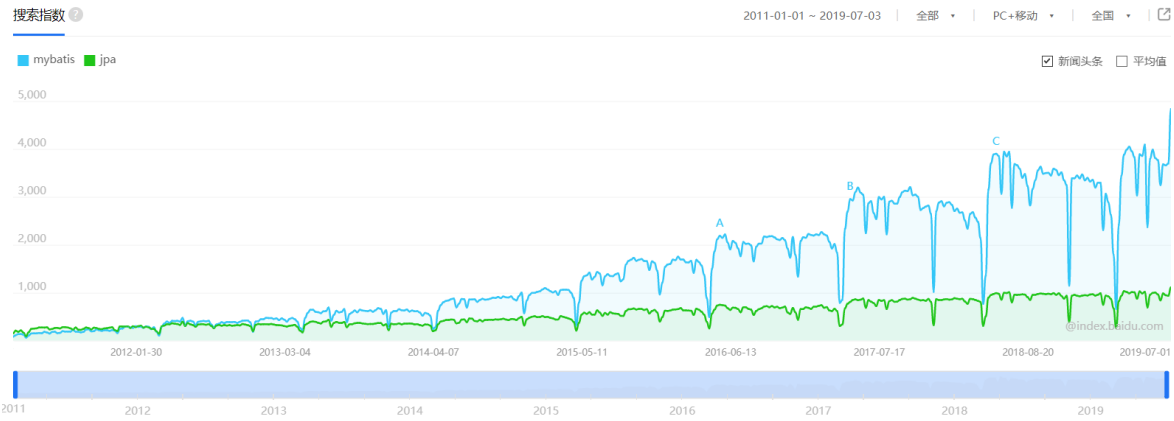
可以理解为JPA规范的再次封装抽象，底层还是使用了Hibernate 的JPA技术实现，引用JPQL (Java Persistence Query Language) 查询语言，属于Spring整个生态体系的一部分。随着Spring Boot和Spring Cloud在市场上的流行，Spring Data JPA也逐渐进入大家的视野，它们组成有机的整体，使用起来比较方便，加快了开发的效率，使开发者不需要关心和配置更多的东西，完全可以沉浸在Spring的完整生态标准实现下。JPA上手简单，开发效率高，对对象的支持比较好，又有很大的灵活性，市场的认可度越来越高。

## 2.ORM框架选型

| 对比项 | Spring Data JPA | Mybatis |
| --- | --- | --- |
| 单表操作方式 | 只需继承，代码量极少，非常方便。而且支持方法名用关键字生成SQL | 可以使用代码生成工具，也很方便，但相对JPA单表弱很多。JPA单表操作简单到令人发指。 |
| 多表关联查询 | 友好，动态SQL使用不够方便，而且SQL和代码耦合到一起 | 非常友好，可以有非常直观的动态SQL |
| 自定义SQL | SQL写在注解里面，写动态SQL有些费劲 | SQL可以写在XML里面，独立管理，动态SQL语法也容易书写与理解 |
| 学习成本 | 略高 | 较低 ,会写SQL就会用 |

由上面的比较可以看出，Spring Data JPA 对开发人员更加友好，单表操作非常方便，多表关联也不麻烦。

mybatis各方面来讲可能都不是非常突出，但是各个方面也都很优秀，使用人数及范围在国内也是更多更广。我个人建议：如果自己研发“小而美”建议使用Spring Data JPA，如果是企业大型项目还是建议选Mybatis。 因为团队技术选型的时候，要考虑学习成本和使用人数，你会用也好用，但别人不会用也不行。下图是百度指数mybatis和jpa关键字的搜索量。可以看出至少在国内mybatis有更多的市场。




# 3.JPA介绍

## 认识JPA

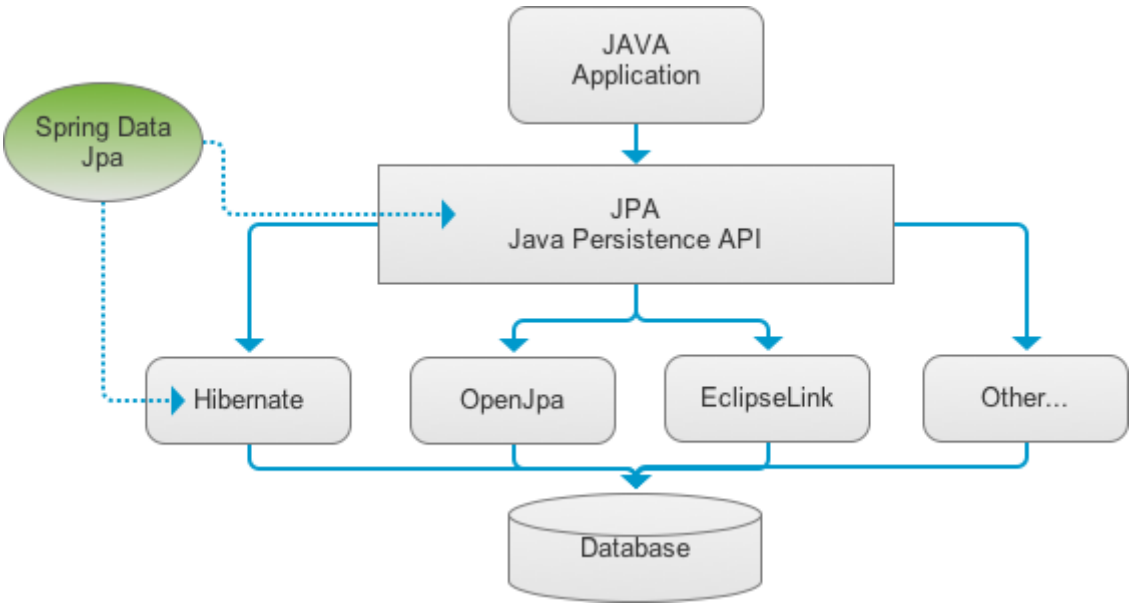
JPA（Java Persistence API）中文名 Java 持久层 API，是 JDK 5.0 注解或 XML 描述对象 - 关系表的映射关系，并将运行期的实体对象持久化到数据库中。

Sun 引入新的 JPA ORM 规范出于两个原因：其一，简化现有 Java EE 和 Java SE 应用开发工作；其二，Sun 希望整合 ORM 技术，实现天下归一。

JPA 包括以下三方面的内容 一套 API 标准，在 javax.persistence 的包下面，用来操作实体对象，执行 CRUD 操作，框架在后台替代我们完成所有的事情，开发者从繁琐的 JDBC 和 SQL 代码中解脱出来。 面向对象的查询语言：Java Persistence Query Language（JPQL），这是持久化操作中很重要的一个方面，通过面向对象而非面向数据库的查询语言查询数据，避免程序的 SQL 语句紧密耦合。

## JPA开源实现

JPA 的宗旨是为 POJO 提供持久化标准规范，由此可见，经过这几年的实践探索，能够脱离容器独立运行，方便开发和测试的理念已经深入人心了。Hibernate 3.2+、TopLink 10.1.3 以及 OpenJPA、QueryDSL 都提供了 JPA 的实现，以及最后的 Spring 的整合 Spring Data JPA。目前互联网公司和传统公司大量使用了 JPA 的开发标准规范




# 4.了解Spring Data

## Spring Data介绍

Spring Data 项目是从 2010 年开发发展起来的，从创立之初 Spring Data 就想提供一个大家熟悉的、一致的、基于 Spring 的数据访问编程模型，同时仍然保留底层数据存储的特殊特性。它可以轻松地让开发者使用数据访问技术包括：关系数据库、非关系数据库（NoSQL）和基于云的数据服务。

Spring Data Common 是 Spring Data 所有模块的公用部分，该项目提供跨 Spring 数据项目的共享基础设施，它包含了技术中立的库接口以及一个坚持 Java 类的元数据模型。

Spring Data 不仅对传统的数据库访问技术：JDBC、Hibernate、JDO、TopLick、JPA、MyBatis 做了很好的支持和扩展、抽象、提供方便的 API，还对 NoSQL 等非关系数据做了很好的支持：MongoDB 、Redis、Apache Solr 等

## Spring Data子项目

主要项目（Main Modules）：

- Spring Data Commons
- Spring Data Gemfire
- Spring Data JPA
- Spring Data KeyValue
- Spring Data LDAP
- Spring Data MongoDB
- Spring Data REST
- Spring Data Redis
- Spring Data for Apache Cassandra
- Spring Data for Apache Solr

社区支持的项目（Community Modules）：

- Spring Data Aerospike
- Spring Data Couchbase
- Spring Data DynamoDB
- Spring Data Elasticsearch
- Spring Data Hazelcast
- Spring Data Jest
- Spring Data Neo4j
- Spring Data Vault

其他（Related Modules）：

- Spring Data JDBC Extensions
- Spring for Apache Hadoop
- Spring Content

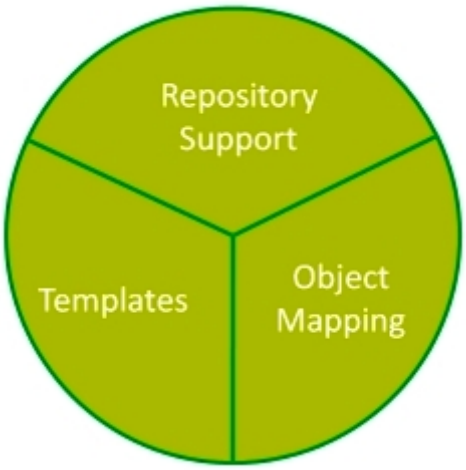
当然了还有许多开源社区做出的许多贡献如 MyBatis 等。

市面上主要的如图所示：

## Spring Data操作的主要特性

Spring Data 项目旨在为大家提供一种通用的编码模式，数据访问对象实现了对物理数据层的抽象，为编写查询方法提供了方便。通过对象映射，实现域对象和持续化存储之间的转换，而模板提供的是对底层存储实体的访问实现，操作上主要有如下特征：

- 提供模板操作，如 Spring Data Redis 和 Spring Data Riak；
- 强大的 Repository 和定制的数据储存对象的抽象映射；
- 对数据访问对象的支持（Auting 等）。




# 5.Spring Data JPA框架

## 简介

**Spring Data JPA** 是 Spring 基于 ORM 框架、JPA 规范的基础上封装的一套 JPA 应用框架，底层使用了 Hibernate 的 JPA 技术实现，可使开发者用极简的代码即可实现对数据的访问和操作。它提供了包括增删改查等在内的常用功能，且易于扩展！学习并使用 Spring Data JPA 可以极大提高开发效率!

## Spring Data JPA的主要类和结构图

### 需要掌握和使用到的类

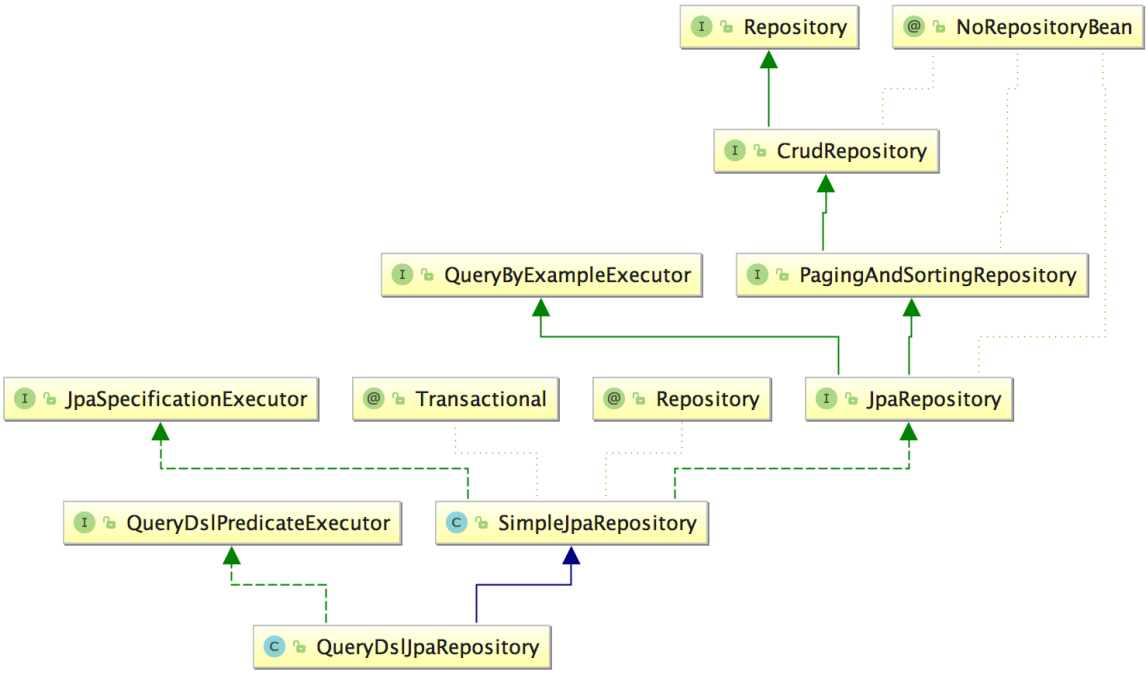
七个大 Repository 接口：

- Repository（org.springframework.data.repository）；
- CrudRepository（org.springframework.data.repository）；
- PagingAndSortingRepository（org.springframework.data.repository）；
- JpaRepository（org.springframework.data.jpa.repository）；
- QueryByExampleExecutor（org.springframework.data.repository.query）；
- JpaSpecificationExecutor（org.springframework.data.jpa.repository）；
- QueryDslPredicateExecutor（org.springframework.data.querydsl）。

两大 Repository 实现类：

- SimpleJpaRepository（org.springframework.data.jpa.repository.support）；
- QueryDslJpaRepository（org.springframework.data.jpa.repository.support）。

### 类的结构关系图




基本上面都是我们要关心的类和接口，先做到大体心中有个数，后面章节会一一做讲解。

**需要了解到的类，真正的 JPA 的底层封装类**

- EntityManager（javax.persistence）；
- EntityManagerImpl（org.hibernate.jpa.internal）。

# 6.Spring Data JPA快速入门

## 环境准备

情景需要：创建spring boot项目，整合spring data jpa项目使用

环境要求：

- JDK8+
- Maven3+
- Intellij IDEA
- Spring Boot2+
- MySQL8+
- Spring Data JPA

## pom.xml

```
<!--spring data jpa依赖-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<!--mysql依赖-->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

注意：spring boot版本为2.1.7

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.7.RELEASE</version>
    <relativePath/>
</parent>
```

## application.properties

```
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/db_jpa?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=root
#创建数据库的方式类型
spring.jpa.hibernate.ddl-auto=update
#显示sql语句
spring.jpa.show-sql=true
```

> `spring.jpa.hibernate.ddl-auto` 是hibernate的配置属性，其主要作用是：自动创建、更新、验证数据库表结构。该参数的几种配置如下：
>
> - `create` ： 每次加载hibernate时都会删除上一次的生成的表，然后根据你的实体类再重新来生成新表，哪怕两次没有任何改变也要这样执行，这就是导致数据库表数据丢失的一个重要原因。
> - `create-drop` ： 每次加载hibernate时根据model类生成表，但是sessionFactory一关闭,表就自动删除。
> - `update` ： 最常用的属性，第一次加载hibernate时根据model类会自动建立起表的结构（前提是先建立好数据库），以后加载hibernate时根据model类自动更新表结构，即使表结构改变了但表中的行仍然存在不会删除以前的行。要注意的是当部署到服务器后，表结构是不会被马上建立起来的，是要等应用第一次运行起来后才会。
> - `validate` ： 每次加载hibernate时，验证创建数据库表结构，只会和数据库中的表进行比较，不会创建新表，但是会插入新值。

## 实体类

```
package com.kazu.springdatajpa01.entity;

import javax.persistence.*;

@Entity//标识是一个实体类
@Table(name="t_user")//表名
public class User {

    @Id//主键
    @GeneratedValue(strategy=GenerationType.IDENTITY)//主键自增类型
    //当属性名与列名一致时，可以省略@Column
    @Column
    private Integer id;
    private String userName;
    private Integer age;
    private String address;
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public Integer getAge() {
        return age;
    }
    public void setAge(Integer age) {
```

```
        this.age = age;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    @Override
    public String toString() {
        return "User [id=" + id + ", userName=" + userName + ", age=" + age + ", address=" + address +
"]";
    }

    public User() {
    }

    public User(String userName, Integer age, String address) {
        this.userName = userName;
        this.age = age;
        this.address = address;
    }
}
```

> 注意：
>
> 1. 属性名若是驼峰命名法，数据库表生成的字段会有下划线_如属性名为userName,列名则是user_name
> 2. 实体类所用注解与Hibernate注解一致

## 接口

> 接口推荐命名方式：实体类名称+Repository
>
> 接口需要继承Spring Data JPA提供的接口

```
package com.kazu.springdatajpa01.dao;

import com.kazu.springdatajpa01.entity.User;
import org.springframework.data.repository.CrudRepository;

/**
 * 用户接口
 */
public interface UserCrudRepository extends CrudRepository<User,Integer> {
}
```

> CrudRepository<T,ID>，其中T为泛型，一般放入持久化类（实体类），ID则指的是主键类型

## 测试类

```
package com.kazu.springdatajpa01;

import com.kazu.springdatajpa01.dao.UserCrudRepository;
import com.kazu.springdatajpa01.entity.User;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import javax.annotation.Resource;

@RunWith(SpringRunner.class)
@SpringBootTest
public class HelloWorldTest {

    @Resource
    private UserCrudRepository userCrudRepository;
```

```
    @Test
    public void test() {
        User user = new User("王五",20,"广州市天河区");
        userCrudRepository.save(user);
    }

}
```

> 具体其他操作方法，后续会逐一讲解