[**第四篇：SpringBoot 整合JPA**](http://blog.csdn.net/forezp/article/details/70545038)

参考：<http://blog.csdn.net/forezp/article/details/70545038>

转载请标明出处：   
<http://blog.csdn.net/forezp/article/details/70545038>   
本文出自[方志朋的博客](http://blog.csdn.net/forezp)

JPA全称Java Persistence API.JPA通过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。

JPA 的目标之一是制定一个可以由很多供应商实现的API，并且开发人员可以编码来实现该API，而不是使用私有供应商特有的API。

JPA是需要Provider来实现其功能的，Hibernate就是JPA Provider中很强的一个，应该说无人能出其右。从功能上来说，JPA就是Hibernate功能的一个子集。

添加相关依赖

添加spring-boot-starter-jdbc依赖：

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa

</artifactId>

</dependency>

添加mysql连接类和连接池类：

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

配置数据源，在application.properties文件配置：

spring:

datasource:

driver-class-name: com.mysql.jdbc.Driver

url: jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=utf8&characterSetResults=utf8

username: root

password: 123456

jpa:

hibernate:

ddl-auto: update # 第一次简表create 后面用update

show-sql: true

注意，如果通过jpa在数据库中建表，将jpa.hibernate,ddl-auto改为create，建完表之后，要改为update,要不然每次重启工程会删除表并新建。

创建实体类

通过@Entity 表明是一个映射的实体类， @Id表明id， @GeneratedValue 字段自动生成

@Entity

public class Account {

@Id

@GeneratedValue

private int id ;

private String name ;

private double money;

... 省略getter setter

}

Dao层

数据访问层，通过编写一个继承自 JpaRepository 的接口就能完成数据访问,其中包含了基本的单表查询的方法，非常的方便。值得注意的是，这个Account 对象名，而不是具体的表名，另外Interger是主键的类型，一般为Integer或者Long

public interface AccountDao extends JpaRepository<Account,Integer> {

}

Web层

在这个栗子中我简略了service层的书写，在实际开发中，不可省略。新写一个controller，写几个restful api来测试数据的访问。

@RestController

@RequestMapping("/account")

public class AccountController {

@Autowired

AccountDao accountDao;

@RequestMapping(value = "/list", method = RequestMethod.GET)

public List<Account> getAccounts() {

return accountDao.findAll();

}

@RequestMapping(value = "/{id}", method = RequestMethod.GET)

public Account getAccountById(@PathVariable("id") int id) {

return accountDao.findOne(id);

}

@RequestMapping(value = "/{id}", method = RequestMethod.PUT)

public String updateAccount(@PathVariable("id") int id, @RequestParam(value = "name", required = true) String name,

@RequestParam(value = "money", required = true) double money) {

Account account = new Account();

account.setMoney(money);

account.setName(name);

account.setId(id);

Account account1 = accountDao.saveAndFlush(account);

return account1.toString();

}

@RequestMapping(value = "", method = RequestMethod.POST)

public String postAccount(@RequestParam(value = "name") String name,

@RequestParam(value = "money") double money) {

Account account = new Account();

account.setMoney(money);

account.setName(name);

Account account1 = accountDao.save(account);

return account1.toString();

}

}

通过postman请求测试，代码已经全部通过测试。

源码下载：<https://github.com/forezp/SpringBootLearning>

参考资料

[accessing-data-jpa](https://spring.io/guides/gs/accessing-data-jpa/)

优秀文章推荐：

* 更多springboot 教程：[springBoot非官方教程 | 文章汇总](http://blog.csdn.net/forezp/article/details/70341818)
* 更多springcoud 教程：[史上最简单的 SpringCloud 教程 | 文章汇总](http://blog.csdn.net/forezp/article/details/70148833)