Spring Boot

- 数据库配置

by万夜

目录

配置JDBC

- 在pom.xml配置maven依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

- Application.properties配置数据源

```
spring.datasource.url=jdbc:mysql://localhost:3306/test
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.max-idle=10
spring.datasource.max-wait=10000
spring.datasource.min-idle=1
spring.datasource.initial-size=1
spring.datasource.validation-query=SELECT 1
spring.datasource.test-on-borrow=false
spring.datasource.test-while-idle=true
spring.datasource.time-between-eviction-runs-millis=18800
```

spring-boot-JDBC实践

- 场景：查询用户列表
- 创建用户实体
- 创建UserService并注入JdbcTemplate
- 实现查询方法，并通过Json输出

配置mybatis

- 在pom.xml配置maven依赖

```
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>1.2.0</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

- Application.properties配置mybatis

```
mybatis.mapper-locations=classpath*:mapper/*Mapper.xml
mybatis.type-aliases-package=com.wanye.entity
```

spring-boot-mybatis实践

- 场景：通过id查询指定用户信息
- 创建mapper接口
- 创建mapper.xml
- 实现查询方法，并通过Json输出

spring-boot-mybatis实践

—UserMapper.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://
mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.wanye.mapper.UserMapper">
    <select id="getUsers" resultType="com.wanye.entity.User">
        SELECT * FROM USER
    </select>

    <select id="getUserById" resultType="com.wanye.entity.User">
        SELECT * FROM USER WHERE id = #{id}
    </select>
</mapper>
```

事务

- 1. 定义：从数据库角度说，就是一组SQL指令，要么全部执行成功，若因为某个原因其中一条指令执行有错误，则撤销先前执行过的所有指令。更简答的说就是：要么全部执行成功，要么撤销不执行。

- 2. java事务：既然事务的概念从数据库而来，那Java事务是什么？之间有什么联系？实际上，一个Java应用系统，如果要操作数据库，则通过JDBC来实现的。增加、修改、删除都是通过相应方法间接来实现的，事务的控制也相应转移到Java程序代码中。因此，数据库操作的事务习惯上就称为Java事务。

- 3. 实现原理(单机事务)：JDBC 事务是用 Connection 对象控制的。JDBC Connection 接口（ java.sql.Connection ）提供了两种事务模式：自动提交和手工提交。 java.sql.Connection 提供了以下控制事务的方法：

- public void setAutoCommit(boolean)
- public boolean getAutoCommit()
- public void commit()
- public void rollback()

- 使用 JDBC 事务界定时，您可以将多个 SQL 语句结合到一个事务中。JDBC 事务的一个缺点是**事务的范围局限于一个数据库连接。**一个 JDBC 事务不能跨越多个数据库。

- 4. JTA（Java Transaction API）事务(多机事务)，通过xa实现，需要驱动支持。

- 5. 容器事务，类似于j2ee容器级别的事务，基本上是基于JTA来实现。

配置事务

- 1. 使用注解 @EnableTransactionManagement 开启事务支持

- 2. 在访问数据库的Service方法上添加注解 @Transactional 便可

spring-boot-事务实践

- 场景：拷贝用户信息，并将原name修改为xiaoming
  - Case1:正常（新增1条记录name=xiaohong，更新1条记录name=xiaoming)
  - Case2:异常（name长度超出var(8)）导致回滚

```
@Transactional
public void copyUser(long id, String name) {
    User user = getUserById(id);
    int i = jdbcTemplate.update("insert INTO USER (name,age,`CREATE`) VALUES (?,?,?)", new Object[]
{user.getName(), user.getAge(), user.getCreate()});
    logger.info("复制成功: i=" + i + " name=" + name + " id=" + id);
    i = jdbcTemplate.update("update USER SET name=? where id=?", new Object[]{name, user.getId()});
    logger.info("更新成功: i=" + i + " name=" + name + " id=" + id);
}
@RequestMapping("/user/cp/{id}/{name}")
public List<User> cpUser(@PathVariable("id") long id, @PathVariable("name") String name) {
    userService.copyUser(id, name);
    return userList();
}
```

总结

- 配置jdbc
- 配置mybatis
- 事务处理
- 关于分页查询
  - 重点是分页算法如何封装
- 关于数据库连接池
  - c3p0/dbcp/tomcat-jdbc/HikariCP
  - spring.datasource.type=com.zaxxer.hikari.HikariDataSource

Spring Boot

- 部署Deploy

by万夜

目录

发布到Tomcat

- 修改pom文件
    - 将pack格式设置为 war

```
<packaging>war</packaging>
```

    - 排除tomcat插件
        - 需要增加servlet-api依赖，否则编译会失败

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>2.5</version>
    <scope>provided</scope>
</dependency>
```

发布到Tomcat

- 修改启动类，继承 SpringBootServletInitializer 并重写 configure 方法

```
@Override
protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {
    return builder.sources(Start.class);
}
```

- 打包部署到容器Tomcat
  - 将war放到webapp目录，注意上下文
  - 将package后目录，放到webapp/ROOT下

总结

- Deploy（发布到Tomcat）

- 本节是Spring Boot系列的最后一节，希望大家有所收获，同时欢迎大家关注公众号“wanye58”，获取课程资料。