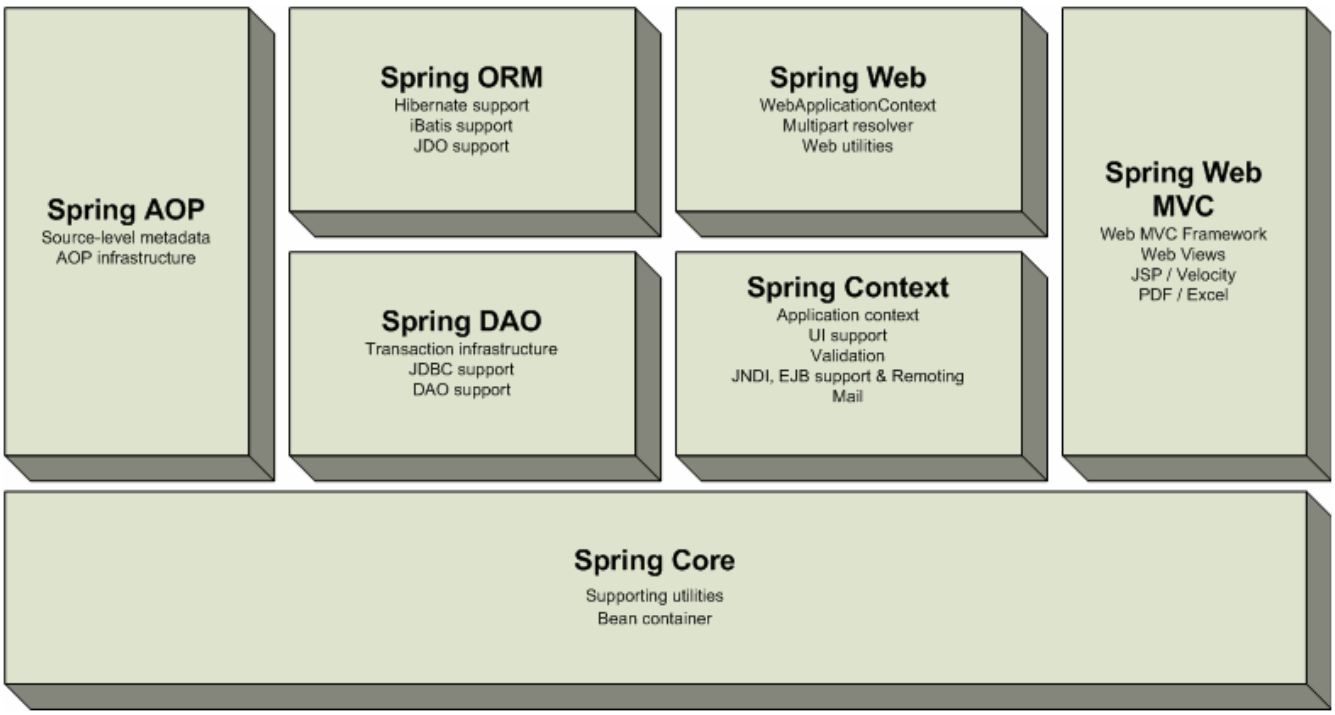
Spring架构

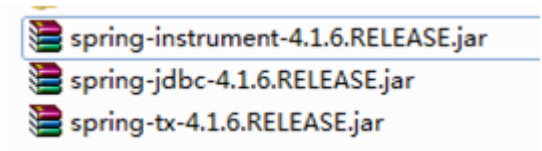


Spring DAO(JDBC模版)

简介

Spring的JDBC模板 为了避免直接使用JDBC而带来的复杂且冗长的代码，Spring提供了一个强有力的模板类---JdbcTemplate来简化JDBC操作。并且，数据源DataSource对象与模板JdbcTemplate对象均可通过Bean的形式定义在配置文件中，充分发挥了依赖注入的威力

需要依赖的JAR：



数据源的配置

使用JDBC模板，首先需要配置好数据源，数据源直接以Bean的形式配置在Spring配置文件中。

由于建立数据库连接是一个非常耗时耗资源的行为，所以通过连接池预先同数据库建立一些连接，放在内存中，应用程序需要建立数据库连接时直接到连接池中申请一个就行，用完后再放回去。

数据源 锁定

本词条由"科普中国"百科科学词条编写与应用工作项目 审核。

数据源是指数据库应用程序所使用的数据库或者数据库服务器。

数据源（Data Source）顾名思义，数据的来源，是提供某种所需要数据的器件或原始媒体。在数据源中存储了所有建立数据库连接的信息。就像通过指定文件名称可以在文件系统中找到文件一样，通过提供正确的数据源名称，你可以找到相应的数据库连接。

数据源连接池Druid

Druid 是阿里巴巴开源平台上的一个项目，整个项目由数据库连接池、插件框架和 SQL 解析器组成。该项目主要是为了扩展 JDBC 的一些限制，可以让程序员实现一些特殊的需求，比如向密钥服务请求凭证、统计 SQL 信息、SQL 性能收集、SQL 注入检查、SQL 翻译等，程序员可以通过定制来实现自己需要的功能。

导入jar

```
mysql-connector-java-5.1.42.jar
druid-1.1.10.jar
//如下是JdbcTemplate需要的jar
spring-instrument-4.3.8.RELEASE.jar
spring-jdbc-4.3.8.RELEASE.jar
spring-tx-4.3.8.RELEASE.jar
```

配置Spring 数据源连接池信息

```
<!-- Durid 数据源相关信息配置 -->
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"
        init-method="init" destroy-method="close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/db_name" />
        <property name="username" value="root" />
        <property name="password" value="root" />
        <!-- 配置初始化大小、最小、最大 -->
        <property name="initialSize" value="1" />
        <property name="minIdle" value="1" />
        <property name="maxActive" value="10" />
        <!-- 配置获取连接等待超时的时间 -->
        <property name="maxWait" value="10000" />
        <!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->
        <property name="timeBetweenEvictionRunsMillis" value="60000" />
        <!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->
        <property name="minEvictableIdleTimeMillis" value="300000" />
        <property name="testWhileIdle" value="true" />
        <!-- 这里建议配置为TRUE，防止取到的连接不可用 -->
```

```
        <property name="testOnBorrow" value="true" />
        <property name="testOnReturn" value="false" />
        <!-- 打开PSCache，并且指定每个连接上PSCache的大小 -->
        <property name="poolPreparedStatements" value="true" />
        <property name="maxPoolPreparedStatementPerConnectionSize"
            value="20" />
        <!-- 这里配置提交方式，默认就是TRUE，可以不用配置 -->
        <property name="defaultAutoCommit" value="true" />
        <!-- 验证连接有效与否的SQL，不同的数据配置不同 -->
        <property name="validationQuery" value="select 1 " />
        <!-- 配置监控统计拦截的filters -->
        <property name="filters" value="stat" />
    </bean>
```

JDBC从属性文件读取数据库连接信息

为了便于维护，可以将数据库连接信息写入到属性文件中，使Spring配置文件从中读取数据。

属性文件名称随意，但一般都是放在src下 如下:jdbc.properties / db.properties

```
#### key=value
jdbc_url=jdbc:mysql://localhost:3306/db_name
jdbc_driver=com.mysql.jdbc.Driver
jdbc_user=root
jdbc_password=root
```

JDBC从属性文件读取数据库连接信息

该属性文件jdbc.properties若要被Spring配置文件读取，其必须在配置文件中进行注册。 注册方式有两种：
（1）方式

（2） context:property-placeholder/方式 该方式要求在Spring配置文件头部加入context的约束，即修改配置文件头

```
<!--
 加载外部配置文件 property
标签中有一个属性location，用于指定属性文件的位置
-->
    <context:property-placeholder  location="classpath:jdbc.properties"/>
```

Spring配置文件从属性文件中读取数据时，需要在的value属性中使用${ }，

将在属性文件中定义的key括起来，以引用指定属性的值

```
<property name="driverClassName" value="${jdbc_driver}" />
<property name="url" value="${jdbc_url}" />
<property name="username" value="${jdbc_user}" />
<property name="password" value="${jdbc_password}" />
```

引入JdbcTemplate:

```
    <!-- 注册JDBC模版对象  -->
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource"></property>
    </bean>
```

完整的application.xml:

```
    <context:component-scan base-package="com.qfjy">
   </context:component-scan>
    <!-- 加载外部配置文件 property -->
    <context:property-placeholder  location="classpath:jdbc.properties"/>

    <!-- Durid 数据源相关信息配置 -->
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"
        init-method="init" destroy-method="close">
        <property name="driverClassName" value="${jdbc_driver}" />
        <property name="url" value="${jdbc_url}" />
        <property name="username" value="${jdbc_user}" />
        <property name="password" value="${jdbc_password}" />
        <!-- 配置初始化大小、最小、最大 -->
        <property name="initialSize" value="1" />
        <property name="minIdle" value="1" />
        <property name="maxActive" value="10" />

        <!-- 配置获取连接等待超时的时间 -->
        <property name="maxWait" value="10000" />

        <!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->
        <property name="timeBetweenEvictionRunsMillis" value="60000" />

        <!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->
        <property name="minEvictableIdleTimeMillis" value="300000" />

        <property name="testWhileIdle" value="true" />

        <!-- 这里建议配置为TRUE，防止取到的连接不可用 -->
        <property name="testOnBorrow" value="true" />
        <property name="testOnReturn" value="false" />

        <!-- 打开PSCache，并且指定每个连接上PSCache的大小 -->
```

```
        <property name="poolPreparedStatements" value="true" />
        <property name="maxPoolPreparedStatementPerConnectionSize"
            value="20" />
        <!-- 这里配置提交方式，默认就是TRUE，可以不用配置 -->
        <property name="defaultAutoCommit" value="true" />
        <!-- 验证连接有效与否的SQL，不同的数据配置不同 -->
        <property name="validationQuery" value="select 1 " />

        <!-- 配置监控统计拦截的filters -->
        <property name="filters" value="stat" />
    </bean>

    <!-- 注册JDBC模版对象  -->
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource"></property>
    </bean>
```

JdbcTemplate 示例

Spring对数据库的操作在jdbc上面做了深层次的封装，使用spring的注入功能，可以把DataSource注册到JdbcTemplate之中。

JdbcTemplate主要提供以下五类方法：

1. query方法及queryForXXX方法：用于执行查询相关语句；
2. update方法及batchUpdate方法：update方法用于执行新增、修改、删除等语句；batchUpdate方法用于执行批处理相关语句；
3. call方法：用于执行存储过程、函数相关语句。
4. execute方法：可以用于执行任何SQL语句，一般用于执行DDL语句；

查询功能

query方法及queryForXXX方法：用于执行查询相关语句

读取单个对象

```
//查询单个对象
public User selectById(int id){
    String sql="select * from user where id=?";
    RowMapper<User> rowMapper=new BeanPropertyRowMapper(User.class);
    User u=jdbcTemplate.queryForObject(sql, rowMapper, id);
    return u;
}
```

读取多个对象

```
public List<User> selectList() {
        String sql="select * from user";
        RowMapper<User> rowMapper=new BeanPropertyRowMapper<>(User.class);
        return jdbcTemplate.query(sql, rowMapper);
}
```

获取某个记录某列 count、max、min、avg、sum等函数返回唯一值

```
public int count() {
        String sql="select count(*) from user";
        return jdbcTemplate.queryForObject(sql,Integer.class);
}
```

【注意】：使用BeanProperytRowMapper要求sql数据查询出来的列和实体属性需要一一对应。如果数据中列名和属性名不一致，在sql语句中需要用as重新取一个别名

修改功能

update方法用于执行新增、修改、删除等语句

新增

```
@Override
    public int insert(User u) {
        String sql="insert into user(uname,upass) values(?,?)";
        return jdbcTemplate.update(sql, u.getUname(),u.getUpass());
    }
```

修改

```
public int update(User u) {
        String sql="update user set uname=? where id=?";
        Object[] objs=new Object[2];
        objs[0]=u.getUname();
        objs[1]=u.getId();
        return jdbcTemplate.update(sql, objs);
    }
```

删除

```
public int deleteById(int id) {
        String sql="delete from user where id=?";
        return  jdbcTemplate.update(sql,id);
}
```

批量插入、更新和删除方法

batchUpdate()

```
public int[] insertBatch(List<User> list) {
        String sql="insert into user(uname,upass) values(?,?)";
        List<Object[]> lists=new ArrayList<Object[]>();
        lists.add(new Object[]{"a","a"});
        lists.add(new Object[]{"b","a"});
        lists.add(new Object[]{"b","a"});
        int[] nums=jdbcTemplate.batchUpdate(sql,lists);
        return nums;
    }
```