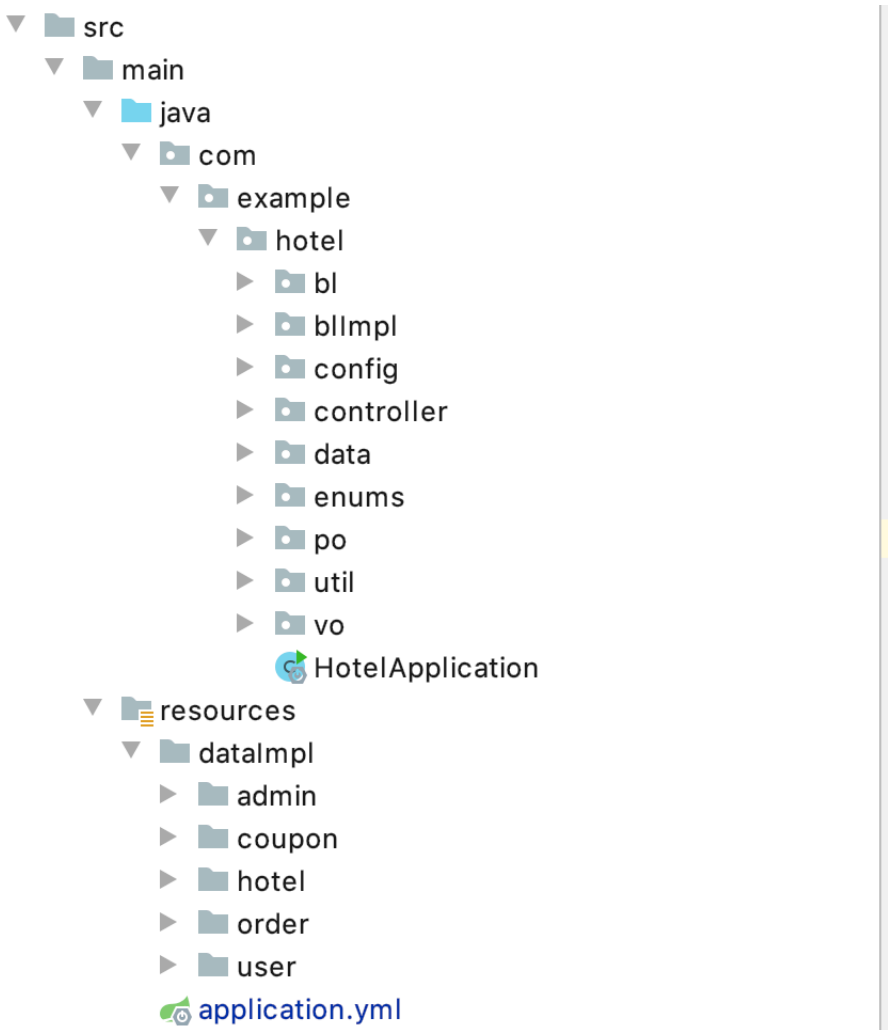
Springboot + Mybatis 后端

项目目录



```
▼ src
  ▼ main
    ▼ java
      ▼ com
        ▼ example
          ▼ hotel
            ▶ bl
            ▶ blImpl
            ▶ config
            ▶ controller
            ▶ data
            ▶ enums
            ▶ po
            ▶ util
            ▶ vo
              HotelApplication
    ▼ resources
      ▼ dataImpl
        ▶ admin
        ▶ coupon
        ▶ hotel
        ▶ order
        ▶ user
        application.yml
```

controller：负责处理REST API请求

bl：业务逻辑层接口

blImpl:业务逻辑层实现

data：数据层接口

po：持久化对象，用于数据层和逻辑层之间数据传递

vo：值对象，用于逻辑层和展示层（前端）数据传递

dataImpl：数据层实现，Mybatis是XML文件

后端调用流程：

controller->bl->blImpl->data->dataImpl

controller

controller.coupon.CouponController

@RestController REST风格API的控制器 @RequestMapping("/api/coupon") 请求的URL

@PostMapping("/hotelTargetMoney") Post请求子路径

couponVO 传给前端的数据

```
package com.example.hotel.controller.coupon;

import com.example.hotel.bl.coupon.CouponService;
import com.example.hotel.vo.CouponVO;
import com.example.hotel.vo.HotelTargetMoneyCouponVO;
import com.example.hotel.vo.OrderVO;
import com.example.hotel.vo.ResponseVO;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/coupon")
public class CouponController {

    @Autowired
    private CouponService couponService;

    @PostMapping("/hotelTargetMoney")
    public ResponseVO addHotelTargetMoneyCoupon(@RequestBody
HotelTargetMoneyCouponVO hotelTargetMoneyCouponVO) {

        CouponVO couponVO =
couponService.addHotelTargetMoneyCoupon(hotelTargetMoneyCouponVO);

        return ResponseVO.buildSuccess(couponVO);
    }

    @GetMapping("/hotelAllCoupons")
    public ResponseVO getHotelAllCoupons(@RequestParam Integer hotelId) {
        return
ResponseVO.buildSuccess(couponService.getHotelAllCoupon(hotelId));
    }

    @GetMapping("/orderMatchCoupons")
    public ResponseVO getOrderMatchCoupons(@RequestParam Integer userId,
                                           @RequestParam Integer hotelId,
                                           @RequestParam Double orderPrice,
                                           @RequestParam Integer roomNum,
                                           @RequestParam String checkIn,
                                           @RequestParam String checkOut) {
```

```
        OrderVO requestOrderVO = new OrderVO();
        requestOrderVO.setUserId(userId);
        requestOrderVO.setHotelId(hotelId);
        requestOrderVO.setPrice(orderPrice);
        requestOrderVO.setRoomNum(roomNum);
        requestOrderVO.setCheckInDate(checkIn);
        requestOrderVO.setCheckOutDate(checkOut);
        return
ResponseVO.buildSuccess(couponService.getMatchOrderCoupon(requestOrderVO));
    }


}
```

bl

Bl.coupon.CouponService

前端输入的参数:

- HotelTargetMoneyCouponVO
- hotelId
- OrderVO 数据层返回的:
- List
- List We should use CouponVO instead of Coupon.

```
package com.example.hotel.bl.coupon;

import com.example.hotel.po.Coupon;
import com.example.hotel.vo.CouponVO;
import com.example.hotel.vo.HotelTargetMoneyCouponVO;
import com.example.hotel.vo.OrderVO;

import java.util.List;

public interface CouponService {
    /**
     * 返回某一订单可用的优惠策略列表
     * @param orderVO
     * @return
     */
    List<Coupon> getMatchOrderCoupon(OrderVO orderVO);  ## Use CouponVO instead
of Coupon

    /**
     * 查看某个酒店提供的所有优惠策略（包括失效的）
     * @param hotelId
```

```
      * @return
      */
    List<Coupon> getHotelAllCoupon(Integer hotelId);

    /**
     * 添加酒店满减优惠策略
     * @param couponVO
     * @return
     */
    CouponVO addHotelTargetMoneyCoupon(HotelTargetMoneyCouponVO couponVO);
}
```

blImpl

blImpl.coupon.CouponServiceImpl.java

```
package com.example.hotel.blImpl.coupon;

import com.example.hotel.bl.coupon.CouponService;
import com.example.hotel.bl.coupon.CouponMatchStrategy;
import com.example.hotel.data.coupon.CouponMapper;
import com.example.hotel.po.Coupon;
import com.example.hotel.vo.CouponVO;
import com.example.hotel.vo.HotelTargetMoneyCouponVO;
import com.example.hotel.vo.OrderVO;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;


@Service
public class CouponServiceImpl implements CouponService {


    private final  TargetMoneyCouponStrategyImpl targetMoneyCouponStrategy;

    private final  TimeCouponStrategyImpl timeCouponStrategy;
    private final CouponMapper couponMapper;

    private static List<CouponMatchStrategy> strategyList = new ArrayList<>();

    @Autowired
    public CouponServiceImpl(TargetMoneyCouponStrategyImpl
targetMoneyCouponStrategy,
                             TimeCouponStrategyImpl timeCouponStrategy,
                             CouponMapper couponMapper) {
```

```java
        this.couponMapper = couponMapper;
        this.targetMoneyCouponStrategy = targetMoneyCouponStrategy;
        this.timeCouponStrategy = timeCouponStrategy;
        strategyList.add(targetMoneyCouponStrategy);
        strategyList.add(timeCouponStrategy);
    }



    @Override
    public List<Coupon> getMatchOrderCoupon(OrderVO orderVO) {

        List<Coupon> hotelCoupons = getHotelAllCoupon(orderVO.getHotelId());

        List<Coupon> availAbleCoupons = new ArrayList<>();

        for (int i = 0; i < hotelCoupons.size(); i++) {
            for (CouponMatchStrategy strategy : strategyList) {
                if (strategy.isMatch(orderVO, hotelCoupons.get(i))) {
                    availAbleCoupons.add(hotelCoupons.get(i));
                }
            }
        }

        return availAbleCoupons;
    }

    @Override
    public List<Coupon> getHotelAllCoupon(Integer hotelId) {
        List<Coupon> hotelCoupons = couponMapper.selectByHotelId(hotelId);
        return hotelCoupons;
    }

    @Override
    public CouponVO addHotelTargetMoneyCoupon(HotelTargetMoneyCouponVO
couponVO) {
        Coupon coupon = new Coupon();
        coupon.setCouponName(couponVO.getName());
        coupon.setDescription(couponVO.getDescription());
        coupon.setCouponType(couponVO.getType());
        coupon.setTargetMoney(couponVO.getTargetMoney());
        coupon.setHotelId(couponVO.getHotelId());
        coupon.setDiscountMoney(couponVO.getDiscountMoney());
        coupon.setStatus(1);
        int result = couponMapper.insertCoupon(coupon);
        couponVO.setId(result);
        return couponVO;
    }
}
```

data

@Mapper @Mapper注解是mybatis的注解，是用来说明这个是一个Mapper，对应的xxxMapper.xml就是来实现这个Mapper。 @Repository @Repository注解是Spring的注解，使用该注解和@Autowired注解把当前类注册成一个bean了

Data.coupon.CouponMapper.java

```
package com.example.hotel.data.coupon;

import com.example.hotel.po.Coupon;
import org.apache.ibatis.annotations.Mapper;
import org.springframework.stereotype.Repository;

import java.util.List;

@Mapper
@Repository
public interface CouponMapper {
    int insertCoupon(Coupon coupon);

    List<Coupon> selectByHotelId(Integer hotelId);
}
```

dataImpl

dataImpl.coupon.CouponMapper.xml

namespace 接口的ID

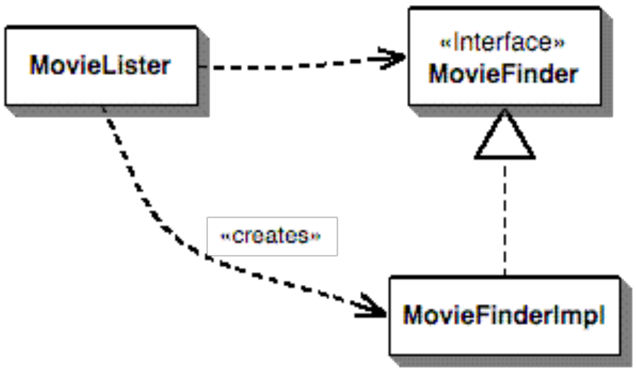
id 方法的ID

```
<?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.example.hotel.data.coupon.CouponMapper">

    <insert id="insertCoupon" parameterType="com.example.hotel.po.Coupon"
            useGeneratedKeys="true" keyProperty="id">
        insert into
coupon(description,couponName,target_money,discount_money,start_time,end_time,hotelId,couponType,discount,status)
        values(#{description},#{couponName},#{targetMoney},#{discountMoney},#{startTime},#{endTime},#{hotelId},#{couponType},#{discount},#{status})
    </insert>  <!--  插入一个优惠券   -->

    <select id="selectByHotelId" resultMap="Coupon">
        select * from Coupon where hotelId=#{hotelId}
    </select>


    <resultMap id="Coupon" type="com.example.hotel.po.Coupon">
        <result column="description" property="description"></result>
        <result column="id" property="id"></result>
        <result column="couponName" property="couponName"></result>
        <result column="hotelId" property="hotelId"></result>
        <result column="couponType" property="couponType"></result>
        <result column="discount" property="discount"></result>
        <result column="status" property="status"></result>
        <result column="target_money" property="targetMoney"></result>
        <result column="discount_money" property="discountMoney"></result>
        <result column="start_time" property="startTime"></result>
        <result column="end_time" property="endTime"></result>
    </resultMap>
</mapper>
```

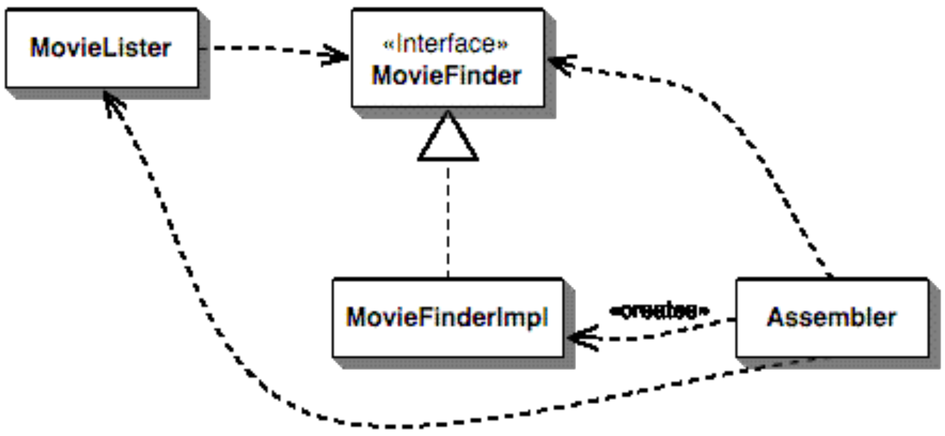
@Autowired

IoC

通过接口调用具体实现的方法，可以降低耦合。但是始终无法消除创建对象时对具体实现的依赖。构造方法没有多态。



通过装配器来完成控制反转 IoC（Inversion of Control）。



DI

依赖注入 DI(Dependency Injection)

class MovieLister...

```
  private MovieFinder finder;
public void setFinder(MovieFinder finder) {
  this.finder = finder;
}
```

Similarly I define a setter for the filename.

class ColonMovieFinder...

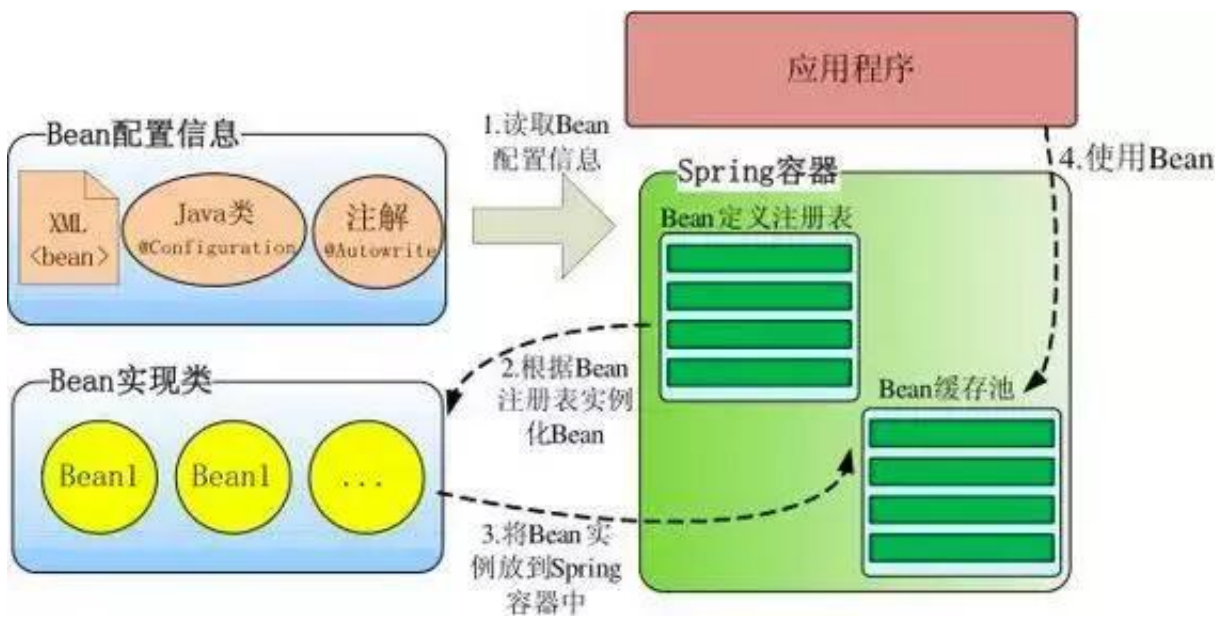
```
  public void setFilename(String filename) {
      this.filename = filename;
  }
```

The third step is to set up the configuration for the files. Spring supports configuration through XML files and also through code, but XML is the expected way to do it.

```
<beans>
    <bean id="MovieLister" class="spring.MovieLister">
        <property name="finder">
            <ref local="MovieFinder"/>
        </property>
    </bean>
    <bean id="MovieFinder" class="spring.ColonMovieFinder">
        <property name="filename">
            <value>movies1.txt</value>
        </property>
    </bean>
</beans>
```

The test then looks like this.

```
public void testWithSpring() throws Exception {
    ApplicationContext ctx = new FileSystemXmlApplicationContext("spring.xml");
    MovieLister lister = (MovieLister) ctx.getBean("MovieLister");
    Movie[] movies = lister.moviesDirectedBy("Sergio Leone");
    assertEquals("Once Upon a Time in the West", movies[0].getTitle());
}
```



ORM和Mybatis

ORM

关系型数据库	面向对象
数据库的表（table）	类（class）
记录（record，行数据）	对象（object）
字段（field）	对象的属性（attribute）

举例来说，下面是一行 SQL 语句。

```
SELECT id, first_name, last_name, phone, birth_date, sex
 FROM persons
 WHERE id = 10
```

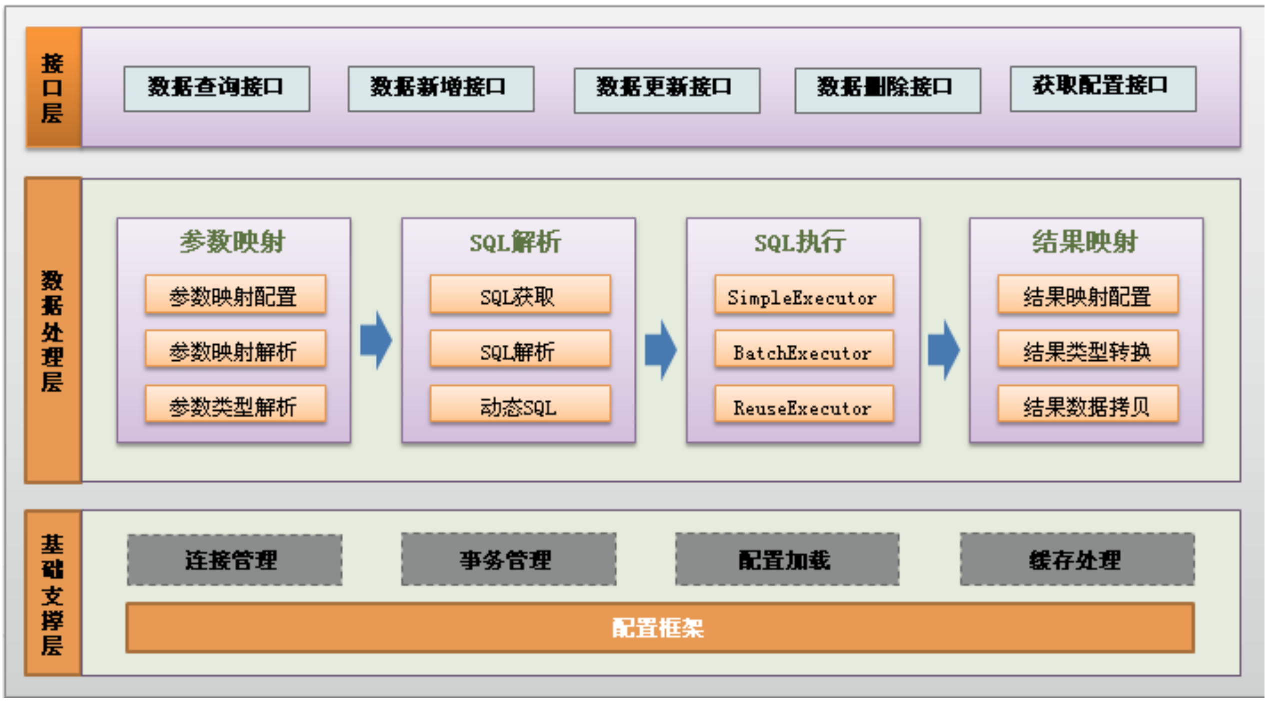
程序直接运行 SQL，操作数据库的写法如下。

```
res = db.execSql(sql);
name = res[0]["FIRST_NAME"];
```

改成 ORM 的写法如下。

```
p = Person.get(10);
name = p.first_name;
```

MyBatis



ORM是Object和Relation之间的映射，包括Object->Relation和Relation->Object两方面。Hibernate是个完整的ORM框架，而MyBatis完成的是Relation->Object，也就是其所说的Data Mapper Framework。

JPA是ORM框架标准，主流的ORM框架都实现了这个标准。MyBatis没有实现JPA，它和ORM框架的设计思路不完全一样。MyBatis是拥抱SQL，而ORM则更靠近面向对象，不建议写SQL，实在要写，则推荐你用框架自带的类SQL代替。MyBatis是SQL映射框架而不是ORM框架，当然ORM和MyBatis都是持久层框架。

最典型的ORM 框架是Hibernate，它是全自动ORM框架，而MyBatis是半自动的。Hibernate完全可以通过对象关系模型实现对数据库的操作，拥有完整的JavaBean对象与数据库的映射结构来自动生成SQL。而MyBatis仅有基本的字段映射，对象数据以及对象实际关系仍然需要通过手写SQL来实现和管理。

pom.xml

```xml
<!--mybatis-->
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>2.1.0</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
```

application.yml

```yaml
spring:
  datasource:
     url: jdbc:mysql://localhost:3306/Hotel?
serverTimezone=CTT&characterEncoding=UTF-8
     username: root
     password: ******
     driver-class-name: com.mysql.cj.jdbc.Driver
     max-active: 200
     max-idle: 20
     min-idle: 10
  thymeleaf:
     cache: false
  jackson:
    time-zone: GMT+8

mybatis:
  mapper-locations: classpath:dataImpl/*/*Mapper.xml
```

参考文献：

https://martinfowler.com/articles/injection.html

https://www.jianshu.com/p/9fe5a3c25ab6