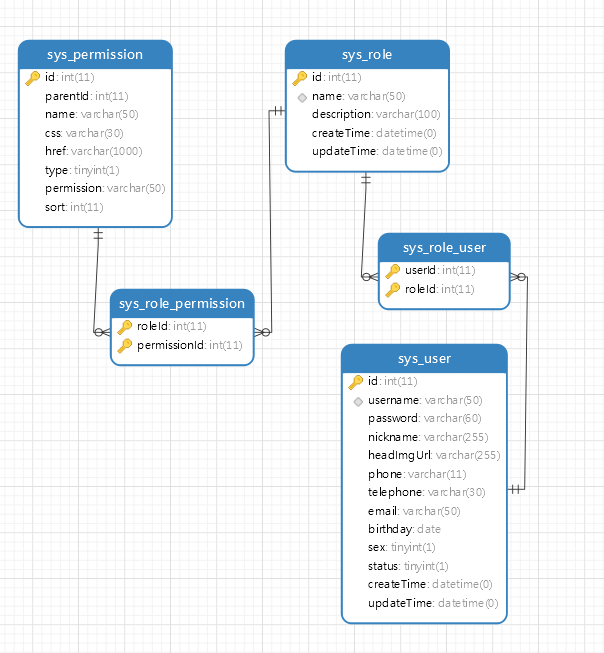
# 项目初步构建

1. 创建项目
2. 配置Logback日志
3. 配置数据源&连接池
4. Mybatis配置
5. 编写一个demo
6. 引入x-admin模板

# 数据库设计

navicat逆向生成模型



权限表 设计方法。

rabc- 用户---角色---权限

# 数据库级联

CASCADE  
在父表上update/delete记录时，同步update/delete掉子表的匹配记录

SET NULL  
在父表上update/delete记录时，将子表上匹配记录的列设为null (要注意子表的外键列不能为not null)    
  
NO ACTION  
如果子表中有匹配的记录,则不允许对父表对应候选键进行update/delete操作    
  
RESTRICT  
同no action, 都是立即检查外键约束  
  
SET NULL  
父表有变更时,子表将外键列设置成一个默认的值 但Innodb不能识别

**NULL、RESTRICT、NO ACTION**

删除：从表记录不存在时，主表才可以删除。删除从表，主表不变

更新：从表记录不存在时，主表才可以更新。更新从表，主表不变

**CASCADE**

删除：删除主表时自动删除从表。删除从表，主表不变

更新：更新主表时自动更新从表。更新从表，主表不变

**SET NULL**

删除：删除主表时自动更新从表值为NULL。删除从表，主表不变

更新：更新主表时自动更新从表值为NULL。更新从表，主表不变

外键约束属性： RESTRICT | CASCADE | SET NULL | NO ACTION  外键的使用需要满足下列的条件：

  1. 两张表必须都是InnoDB表，并且它们没有临时表。

  2. 建立外键关系的对应列必须具有相似的InnoDB内部数据类型。

  3. 建立外键关系的对应列必须建立了索引。

  4. 假如显式的给出了CONSTRAINT symbol，那symbol在数据库中必须是唯一的。假如没有显式的给出，InnoDB会自动的创建。

  如果子表试图创建一个在父表中不存在的外键值，InnoDB会拒绝任何INSERT或UPDATE操作。如果父表试图UPDATE或者DELETE任何子 表中存在或匹配的外键值，最终动作取决于外键约束定义中的ON UPDATE和ON DELETE选项。InnoDB支持5种不同的动作，如果没有指定ON DELETE或者ON UPDATE，默认的动作为RESTRICT:

  1. CASCADE: 从父表中删除或更新对应的行，同时自动的删除或更新自表中匹配的行。ON DELETE CANSCADE和ON UPDATE CANSCADE都被InnoDB所支持。

  2. SET NULL: 从父表中删除或更新对应的行，同时将子表中的外键列设为空。注意，这些在外键列没有被设为NOT NULL时才有效。ON DELETE SET NULL和ON UPDATE SET SET NULL都被InnoDB所支持。

  3. NO ACTION: InnoDB拒绝删除或者更新父表。

  4. RESTRICT: 拒绝删除或者更新父表。指定RESTRICT（或者NO ACTION）和忽略ON DELETE或者ON UPDATE选项的效果是一样的。

  5. SET DEFAULT: InnoDB目前不支持。

  外键约束使用最多的两种情况无外乎：

  1）父表更新时子表也更新，父表删除时如果子表有匹配的项，删除失败；

  2）父表更新时子表也更新，父表删除时子表匹配的项也删除。

多对多 多对一删除用

  前一种情况，在外键定义中，我们使用ON UPDATE CASCADE ON DELETE RESTRICT；后一种情况，可以使用ON UPDATE CASCADE ON DELETE CASCADE。

当执行外键检查之时，InnoDB对它照看着的子或父记录设置共享的行级锁。InnoDB立即检查外键约束，检查不对事务提交延迟。

要使得对有外键关系的表重新载入转储文件变得更容易，mysqldump自动在转储输出中包括一个语句设置FOREIGN\_KEY\_CHECKS为0。这避免在转储被重新装载之时，与不得不被以特别顺序重新装载的表相关的问题。也可以手动设置这个变量：

mysql> SET FOREIGN\_KEY\_CHECKS = 0;

mysql> SOURCE dump\_file\_name;

mysql> SET FOREIGN\_KEY\_CHECKS = 1;

　　如果转储文件包含对外键是不正确顺序的表，这就以任何顺序导入该表。这样也加快导入操作。设置FOREIGN\_KEY\_CHECKS为0，对于在LOAD DATA和ALTER TABLE操作中忽略外键限制也是非常有用的。

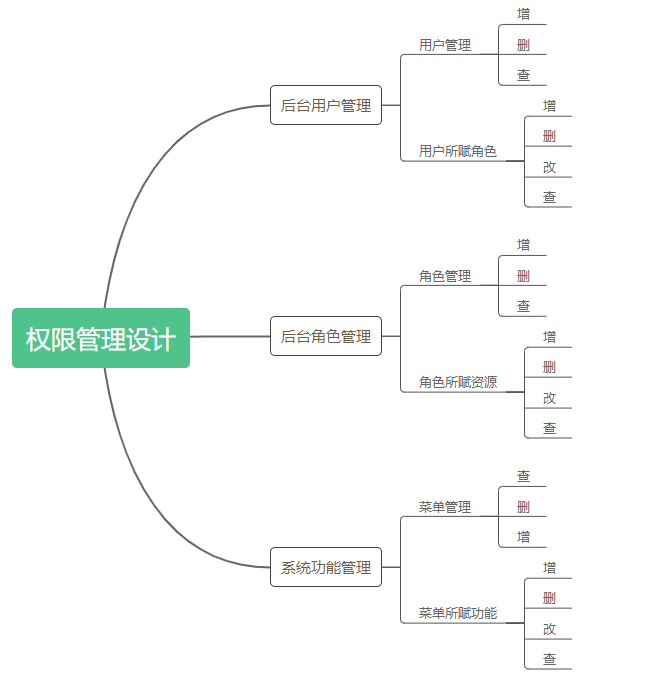
InnoDB不允许你删除一个被FOREIGN KEY表约束引用的表，除非你做设置SET FOREIGN\_KEY\_CHECKS=0。当你移除一个表的时候，在它的创建语句里定义的约束也被移除。

如果你重新创建一个被移除的表，它必须有一个遵从于也引用它的外键约束的定义。它必须有正确的列名和类型，并且如前所述，它必须对被引用的键有索引。如果这些不被满足，MySQL返回错误号1005 并在错误信息字符串中指向errno 150。

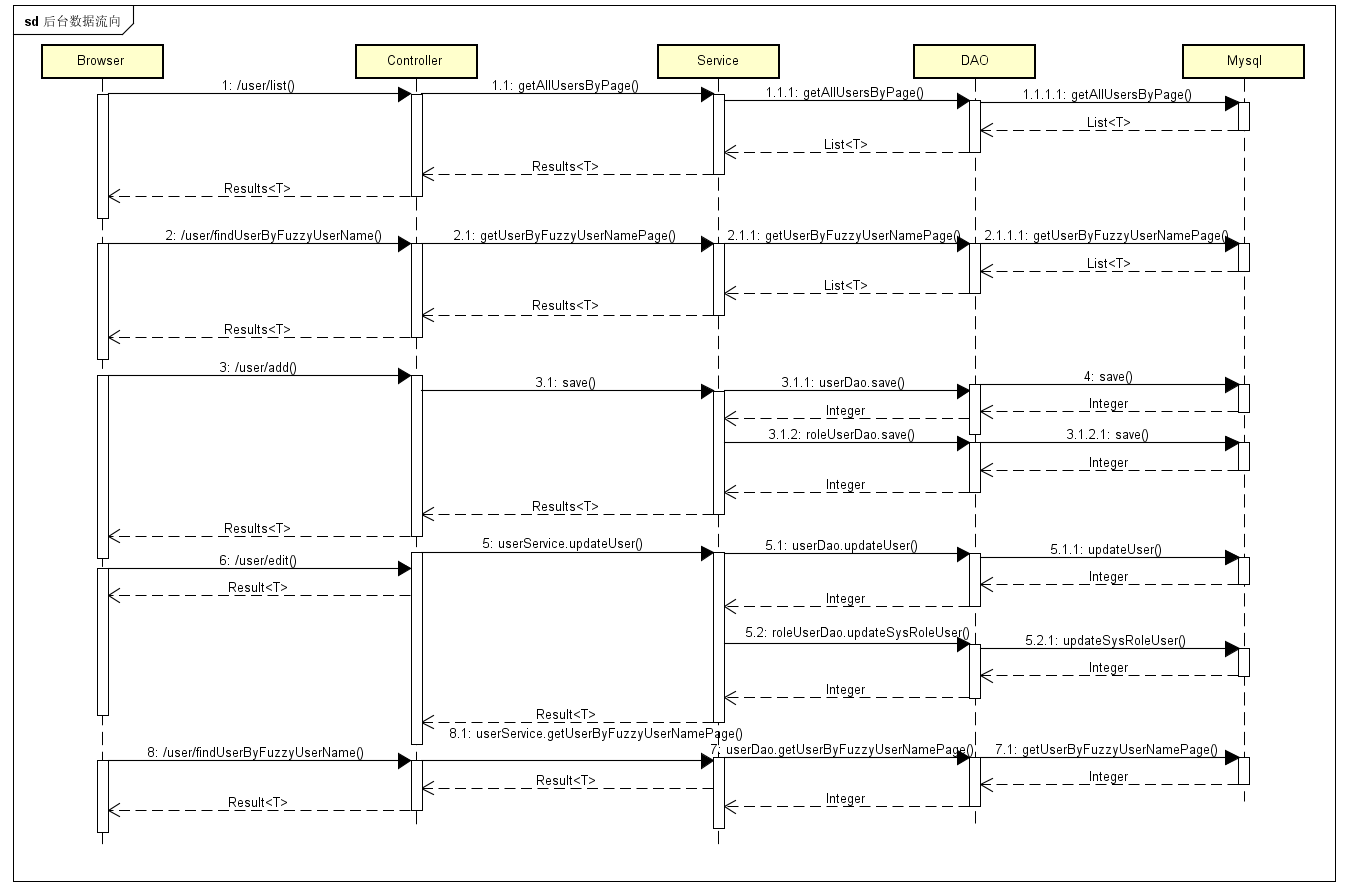
多对多 多对一 删除 级联删除 cascade

NULL、RESTRICT、NO ACTION  
删除：从表记录不存在时，主表才可以删除。删除从表，主表不变  
更新：从表记录不存在时，主表才可以更新。更新从表，主表不变  
   
CASCADE  
删除：删除主表时自动删除从表。删除从表，主表不变  
更新：更新主表时自动更新从表。更新从表，主表不变  
   
SET NULL  
删除：删除主表时自动更新从表值为NULL。删除从表，主表不变  
更新：更新主表时自动更新从表值为NULL。更新从表，主表不变  
  
sys\_role\_permission  
删除用restrict  
更新cascade  
sys\_role\_user 都是restrict

# 权限管理设计



# 用户列表功能设计



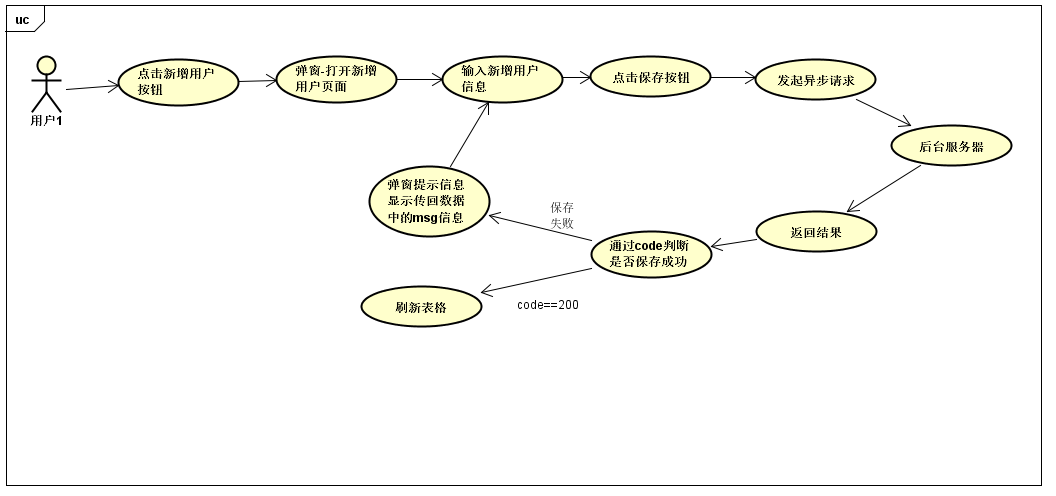
# 分页

使用场景:

• 客户端的问题：加载太慢，不便于操作 • 服务端的问题：内存溢出，性能



# 用户新增



# 角色管理

角色管理的crud

树形结构实现：ztree 之后需要自己找个比较好看的类似ztree的东西

ztree是jquery来实现的 递归调用

# Mybatis dao sql

<http://www.mybatis.cn/archives/739.html>

# Mybatis mapper sql

<https://blog.csdn.net/Dennis_Wu_/article/details/72140740>

# serialVersionUID

不写serialVersionUID就没有吗  
即使不写， jdk序列化时默认也会生成这个id, 反编译.class文件你也看不到这个值  
  
小结一下：  
  
1 值的作用：  
  
用于判断序列化文件是否已经失效（过期）。  
  
序列化的时候会把这个ID写到文件里。  
  
读的时候会把这个ID和代码里的ID比较，如果不一致，表示文件里的已经失效。（will result in an InvalidClassException.）  
  
2 值写为多少：  
  
你可以写为1L，也可以让ECLIPSE帮你生成一个。  
  
3 不写会有什么问题？  
  
不写的话，序列话的时候，JVM会帮你动态的生成一个。这个动态生成的算法可能在不同的虚拟机里不一样，也就是不同的环境下生成的可能不同。这就会有一个问题，你序列化产生的文件，别人读的时候会 InvalidClassException。  
  
所以为了避免这个问题，这个值要写！  
 ————————————————  
 版权声明：本文为CSDN博主「Zonson9999」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。  
 原文链接：https://blog.csdn.net/wuzhong8809/article/details/83416579

# lombok @EqualsAndHashCode 注解的影响

[官方文档：@EqualsAndHashCode](https://projectlombok.org/features/EqualsAndHashCode.html)

原文中提到的大致有以下几点：

1. 此注解会生成equals(Object other) 和 hashCode()方法。

2. 它默认使用非静态，非瞬态的属性

3. 可通过参数exclude排除一些属性

4. 可通过参数of指定仅使用哪些属性

5. 它默认仅使用该类中定义的属性且不调用父类的方法

6. 可通过callSuper=true解决上一点问题。让其生成的方法中调用父类的方法。

另：@Data相当于@Getter @Setter @RequiredArgsConstructor @ToString @EqualsAndHashCode这5个注解的合集。

通过官方文档，可以得知，当使用@Data注解时，则有了@EqualsAndHashCode注解，那么就会在此类中存在equals(Object other) 和 hashCode()方法，且不会使用父类的属性，这就导致了可能的问题。

比如，有多个类有相同的部分属性，把它们定义到父类中，恰好id（数据库主键）也在父类中，那么就会存在部分对象在比较时，它们并不相等，却因为lombok自动生成的equals(Object other) 和 hashCode()方法判定为相等，从而导致出错。

修复此问题的方法很简单：

1. 使用@Getter @Setter @ToString代替@Data并且自定义equals(Object other) 和 hashCode()方法，比如有些类只需要判断主键id是否相等即足矣。

2. 或者使用在使用@Data时同时加上@EqualsAndHashCode(callSuper=true)注解。

# Spingboot中路由传参

<https://www.cnblogs.com/appium/p/12056234.html>

1.介绍几种如何处理url中的参数的注解

 @PathVaribale  获取url中的数据

 @RequestParam  获取请求参数的值

 @GetMapping  组合注解，是 @RequestMapping(method = RequestMethod.GET) 的缩写

(1)PathVaribale 获取url中的数据

　　看一个例子，如果我们需要获取Url=localhost:8080/hello/id中的id值，实现代码如下：

@RestController

public class HelloController {

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

public String sayHello(@PathVariable("id") Integer id,@PathVariable("name") String name){

return "id:"+id+" name:"+name;

}

}

同样，如果我们需要在url有多个参数需要获取，则如下代码所示来做就可以了。

@RestController

public class HelloController {

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

public String sayHello(@PathVariable("id") Integer id,@PathVariable("name") String name){

return "id:"+id+" name:"+name;

}

}

　以上，通过 @PathVariable 注解来获取URL中的参数时的前提条件是我们知道url的格式时怎么样的。  
　　只有知道url的格式，我们才能在指定的方法上通过相同的格式获取相应位置的参数值。  
　　一般情况下，url的格式为： localhost:8080/hello?id=98 ,这种情况下该如何来获取其id值呢,这就需要借助于 @RequestParam 来完成了

2.@RequestParam 获取请求参数的值

　　例如：

@RestController

public class HelloController {

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

public String sayHello(@RequestParam("id") Integer id){

return "id:"+id;

}

}@RequestParam 注解给我们提供了这种解决方案，即允许用户不输入id时，使用默认值，具体代码如下：

@RestController

public class HelloController {

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

//required=false 表示url中可以不穿入id参数，此时就使用默认参数

public String sayHello(@RequestParam(value="id",required = false,defaultValue = "1") Integer id){

return "id:"+id;

}

}

如果在url中有多个参数，即类似于 localhost:8080/hello?id=98&&name=helloworld 这样的url，同样可以这样来处理。具体代码如下：

@RestController

public class HelloController {

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

public String sayHello(@RequestParam("id") Integer id,@RequestParam("name") String name){

return "id:"+id+ " name:"+name;

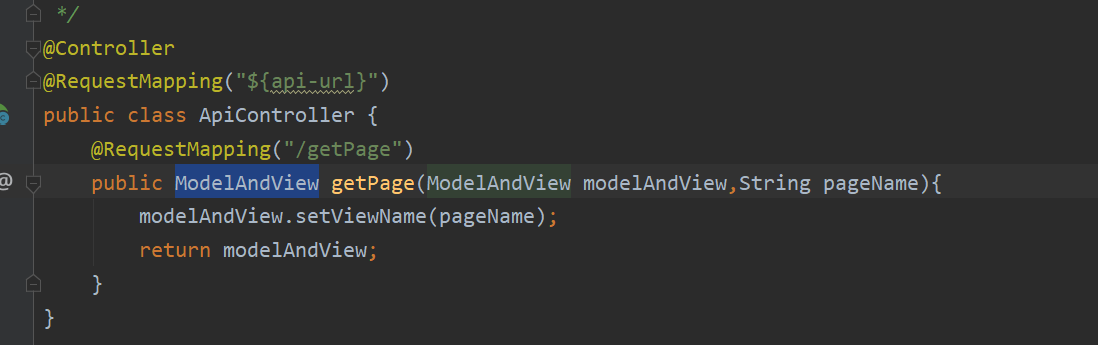
}

}

# ModelA

# ndView

返回视图，用来请求页面 api



路由问号传参，不用param直接捕捉一个参数，单个参数可以这样