1、数据库表

（1）、user_role　用户角色关联表

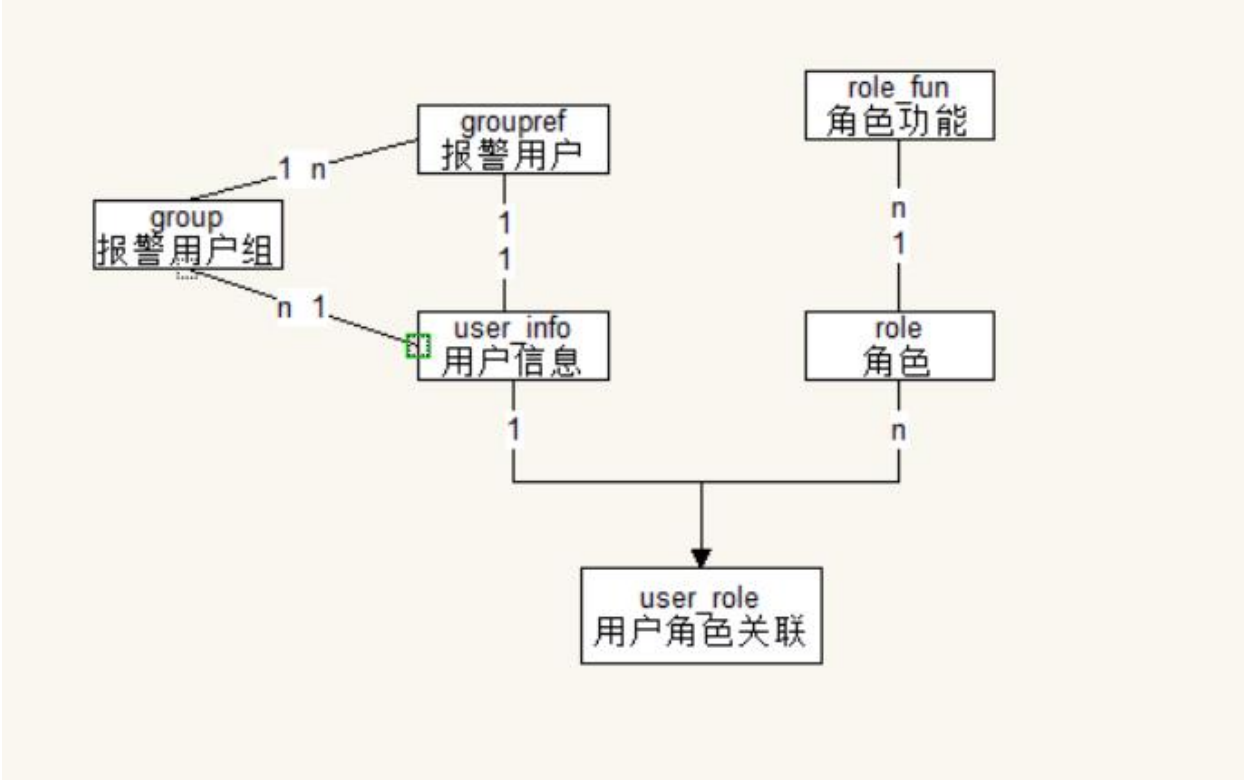
（2）、user_info　用户信息表

（3）、role_fun　　用户功能表

（4）、role　　角色表

（5）、alarm_receiver_group_ref　报警用户

（6）、alarm_receiver_group　报警用户组



关于用户表对应角色功能表生成第三张表的解释：

用户是具体到某一个账户个体，角色是某一类账户的集合。

比如说仓库管理员，这是一个角色，他们管理着仓库；

但是可能公司里有好几个仓库管理员，他们是不同的用户；

在数据库里，角色代表一系列权限的集合，如果将某个角色分配给某个用户，则这个用户就拥有了这一系列的权限。

这样做的好处是不必为单个用户分配不同的权限；假如公司新招库管员一名，直接赋予他库管员的角色，他就有了管理仓库的一系列权限，如果这个人要调换到销售部门去，那么直接给他换个角色就解决了权限的问题。

数据库下同理，有些用户只能查询某些表，而有些用户又只能执行SP，这种权限就需要角色来管理。

2. 用户

应用系统的具体操作者，用户可以自己拥有权限信息，可以归属于0～n个角色，可属于0～n个组。他的权限集是自身具有的权限、所属的各角色具有的权限、所属的各组具有的权限的合集。它与权限、角色、组之间的关系都是n对n的关系。

3. 角色

为了对许多拥有相似权限的用户进行分类管理，定义了角色的概念，例如系统管理员、管理员、用户、访客等角色。角色具有上下级关系，可以形成树状视图，父级角色的权限是自身及它的所有子角色的权限的综合。父级角色的用户、父级角色的组同理可推。

4. 组

为了更好地管理用户，对用户进行分组归类，简称为用户分组。组也具有上下级关系，可以形成树状视图。在实际情况中，我们知道，组也可以具有自己的角色信息、权限信息。这让我想到我们的QQ用户群，一个群可以有多个用户，一个用户也可以加入多个群。每个群具有自己的权限信息。例如查看群共享。QQ群也可以具有自己的角色信息，例如普通群、高级群等。

5、 关于数据库中的关系表的作用

1 个用户对应 1 个角色（表）

（1）、这是为了实现n：n 的关系

（2）、避免字段冗余，加入uid 为用户表主键， cid为课程表主键，我们需要用户时，查询用户表即可，需要课程表查询课程即可，需要查看哪个用户选了哪几门课程时，就可以通过uid关联到cid来查询，如果写在一块的话，字段太多，造成冗余

6、springBoot多环境（dev， test，prod）配置

properties配置格式：

在spring Boot 中多环境配置文件需要满足application-{profile}.properties的格式，其中{profile} 对应你的环境标识：

application-dev.properties：开发环境

application-test.properties：测试环境

application-prod.properties：生产环境

至于哪个具体的配置文件会被加载，需要在application.properties文件中通过spring,profiles.aciton属性来设置，其值对应{profile} 值

如：spring.profiles.action = dev 就会加载application-dev.properties配置文件内容

7、springBoot中常用注解（@PathVaribale 、 @RequestParam 、 @GetMapping)

@PathVaribale：获取Url中的数据

@RequestParam：获取请求参数的值

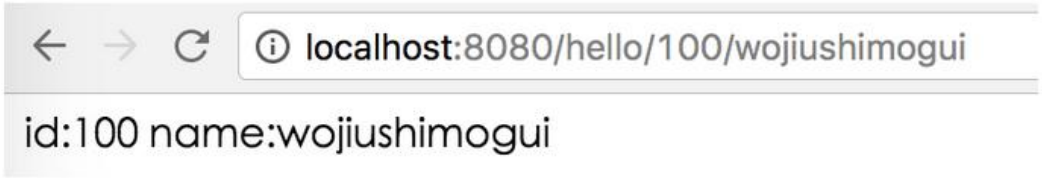
@GetMapping：组合注解，是@RequestMapping(method = RequestMathod.GET)

关于@PathVarlbale

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;
    }
}
```

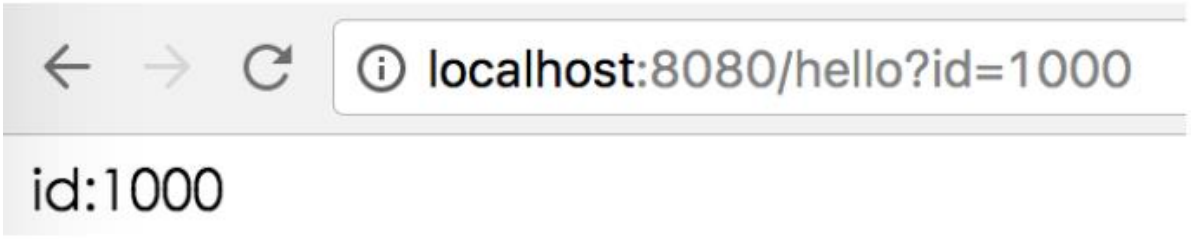


关于@RequestParam

```java
@RestController
public class HelloController {

    @RequestMapping(value="/hello",method= RequestMethod.GET)
    public String sayHello(@RequestParam("id") Integer id){
        return "id:"+id;
    }
}
```

在浏览器中输入地址：localhost:8080/hello?id=1000，可以看到如下的结果：



@GetMapping 组合注解

@GetMapping是一个组合注解，是@RequestMapping(method = RequestMethod.GET)的缩写。该注解将HTTP Get 映射到 特定的处理方法上。

即可以使用@GetMapping(value = “/hello”)来代替@RequestMapping(value=”/hello”,method= RequestMethod.GET)。即可以让我们精简代码。

例子

```java
@RestController
public class HelloController {
    //@RequestMapping(value="/hello",method= RequestMethod.GET)
    @GetMapping(value = "/hello")
    //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;
    }
}
```