rose手册计划

- 目标：光大rose在国内java行业的使用，降低java入门。
- 人人网、糯米网释出的、开源的高效Java web开发框架。在小米米聊服务端再次被验证和使用。一个从零开始的创业公司，在大家技术背景不一的情况下，rose很简单快速地传达到了大家中间。本手册致力于让php开发人员也能快速使用上java开发高性能服务。

- 如果在阅读过程中有任何疑问，欢迎来信咨询：czhttp@gmail.com (或者在 http://www.54chen.com 留言)，将会第一时间得到答复。

- 小米科技 bmw 团队荣誉出品。（BasicMiliaoWare）

章节计划

rose手册第一章：入门指引

1.1 简介：

人人网、糯米网释出的、开源的高效Java web开发框架。在小米米聊服务端再次被验证和使用。一个从零开始的创业公司，在大家技术背景不一的情况下，rose很简单快速地传达到了大家中间。本手册致力于让php开发人员也能快速使用上java开发高性能服务。

1）rose是什么？

- 基于IoC容器 (使用Spring 2.5.6).
- 收集最佳实践，形成规范和惯例，引导按规范惯例，简便开发.
- 收集通用功能，形成一些可使用的组件，提高生产效率.
- 特性的插拔，使用基于组合而非继承的设计.
- 提供可扩展的点，保持框架的可扩展性.
- 注重使用简易性的同时，注重内部代码设计和实现.

如果你是一个创业公司在选择php还是java，同时如果你的团队有一个人写过一年java其他人都没写过。如果你想选择一个更加大型的系统框架，请使用rose，它收集了来自人人网、糯米网、小米科技的众多工程师的经验，你可以免费拥有这些。

2）rose能做什么？

- 初级rose用户：
    - rose可以用来完成一个网站。
- 中级rose用户：
    - rose可以用来完成一个大型网站，它提供的jade功能使得你的项目可以快速开发，自然切入连接池；它提供的portal功能，可以将一个网页分多个线程发起向DB的请求，节省用户的时间；它提供的pipe功能类似facebook的bigpipe，让前端加速，与此同时还有portal多线程的优势。
- 高级rose用户：
    - rose可以自由加入spring任何特性，比如定时执行（去TM的crontab）；比如拦截器做统一权限控制。可以自由配置主库从库，分表规则。配置thrift、zookeeper可以得到牛B的高可用性高性能服务集群。

1.2 简明教程：

*下面开始来进入rose框架下的开发。只需要有一个感性的认识即可。下一章里会专门详细的手把手教你搭建hello world项目。

1）需要些什么？

- 提前学习什么是maven：简单地说，maven是个build工具，用一个pom.xml来定义项目的依赖。通过一个命令mvn，可以自由地build compile（知道make吧，类似，或者类似ant）。
- 也许还需要一个nexus，用来搭建自己的maven仓库（这些都是门槛啊，知道为什么java用的人在全球多，而在中国php的人似乎更多，因为我们的基础设施太落后了）。nexus的作用是配合maven工作。（54chen正在向sonatype申请将rose项目push到sonatype的官方库中，成功后这将省略掉这一步）

- 然后需要在你的项目的pom文件中添加：

```
<dependency>
    <groupId>com.54chen</groupId>
    <artifactId>paoding-rose</artifactId>
    <version>1.0</version>
</dependency>
<dependency>
    <groupId>com.54chen</groupId>
    <artifactId>paoding-rose-jade</artifactId>
    <version>1.1</version>
</dependency>
<dependency>
    <groupId>com.54chen</groupId>
    <artifactId>paoding-rose-scanning</artifactId>
    <version>1.0</version>
</dependency>
```

- 这是三个最基础的框架包。

2）一个controller长什么样？

```
@Path("/")
public class TestController {
    @Get("hello")
    public String test(){
        return "@a";
    }
}
```

- http://localhost/hello

- 将会返回:a。就是这么简单。

下一节预告：rose手册第二章：配置与使用

rose手册第二章：配置与使用

2.1 基础环境

- 普通的pc机，del 380
- ubuntu 10.04基本不升级
- java version "1.6.0_29"
- eclipse
- m2clipse
- 茶一杯

2.2 maven简介

- maven是基于项目对象模型(POM)，可以通过一小段描述信息来管理项目的构建，报告和文档的软件项目管理工具。如果你已经有十次输入同样的Ant targets来编译你的代码、jar或者war、生成javadocs，你一定会自问，是否有一个重复性更少却能同样完成该工作的方法。Maven便提供了这样一种选择，将你的注意力从作业层转移到项目管理层。Maven项目已经能够知道如何构建和捆绑代码，运行测试，生成文档并宿主项目网页。
- maven对一个项目进入了固定的默认目录定义：
    - src/main/java 写主要的java实现
    - src/main/resources 写主要的配置文件
    - src/test/java 写test case
    - src/test/resources 写test case所需要的配置文件

- src/main/webapp [war项目特有]web项目的对外目录
- src/main/webapp/WEB-INF [war项目特有]web项目配置web.xml目录

2.3 项目建立

- 打开eclipse(需要提前安装好m2clipse插件)
- new -> other -> maven -> maven project
- create a simple project
- next
- group id:com.54chen
- artifact id:rose-example
- packaging: war
- finished

2.4 基础配置三步走

1）点火：基础的pom文件

打开2.3建立好的项目，打开pom.xml，添加下面的段落到project中：

```
<dependencies>
    <dependency>
        <groupId>com.54chen</groupId>
        <artifactId>paoding-rose-scanning</artifactId>
        <version>1.0</version>
    </dependency>

    <dependency>
        <groupId>com.54chen</groupId>
        <artifactId>paoding-rose</artifactId>
        <version>1.0</version>
    </dependency>

    <dependency>
        <groupId>com.54chen</groupId>
        <artifactId>paoding-rose-portal</artifactId>
        <version>1.0</version>
    </dependency>

    <dependency>
        <groupId>com.54chen</groupId>
        <artifactId>paoding-rose-jade</artifactId>
        <version>1.1</version>
    </dependency>
</dependencies>
```

上述是rose环境最基础的依赖包。再添加一点常见的编译设置：

```
<build>
    <resources>
        <resource>
            <directory>src/main/resources</directory>
            <includes>
                <include>**/*.*</include>
            </includes>
        </resource>
    </resources>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
            <version>2.0.2</version>
            <configuration>
                <webResources>
                    <resource>
                        <targetPath>WEB-INF</targetPath>
                        <filtering>true</filtering>
                        <directory>src/main/resources</directory>
                        <includes>
                            <include>**/*.xml</include>
                            <include>**/*.properties</include>
                        </includes>
                        <targetPath>WEB-INF</targetPath>
                    </resource>
```

```
                    </webResources>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.6</source>
                    <target>1.6</target>
                    <fork>true</fork>
                    <verbose>true</verbose>
                    <encoding>UTF-8</encoding>
                    <compilerArguments>
                        <sourcepath>
                            ${project.basedir}/src/main/java
                        </sourcepath>
                    </compilerArguments>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <configuration>
                    <!-- 忽略测试 -->
                    <skip>true</skip>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

上述是编译设置，也是放在project段落里。

2）松离合：必不可少的web.xml

在src/main/webapp/WEB-INF文件夹下建立web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    id="WebApp_ID" version="2.5">
    <display-name>rose-example</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.htm</welcome-file>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>


    <context-param>
        <param-name>log4jConfigLocation</param-name>
        <param-value>/WEB-INF/log4j.xml</param-value>
    </context-param>

    <listener>
        <listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>
    </listener>
    <filter>
        <filter-name>roseFilter</filter-name>
        <filter-class>net.paoding.rose.RoseFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>roseFilter</filter-name>
        <url-pattern>/*</url-pattern>
        <dispatcher>REQUEST</dispatcher>
        <dispatcher>FORWARD</dispatcher>
        <dispatcher>INCLUDE</dispatcher>
    </filter-mapping>
</web-app>
```

3）踩油门：applicationContext.xml

src/main/resources/applicationContext.xml是spring环境的油门，所有包的扫描和启动都在这里定义：

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-2.5.xsd"
    default-lazy-init="true">

    <!-- 自动扫描 -->
    <context:annotation-config />
    <context:component-scan base-package="com.chen">
    </context:component-scan>

</beans>
```

2.5 hello world

- 在src/main/java上右键 -> new -> package -> name: com.chen
- 在com.chen上右键 -> new -> package -> com.chen.controllers [controllers是rose框架默认的加载controller的package name]
- 在com.chen.controllers上右键 -> new -> class -> HelloController [*Controller是rose框架默认的controller层的class后缀]
- 打开HelloController这个类
- 在public class HelloController添加注解@Path("") [Path注解是rose框架提供的标识每个controller的对外访问时的基础路径]
- 在HelloController中添加方法

```java
/**
 * @author 54chen(陈臻) [chenzhen@xiaomi.com czhttp@gmail.com]
 * @since 2012-4-10 上午11:14:46
 */
package com.chen.controllers;

import net.paoding.rose.web.annotation.Path;
import net.paoding.rose.web.annotation.rest.Get;

@Path("")
public class HelloController {

    @Get("")
    public String index() {
        return "@hello world";
    }
}
```

- [Get注解是rose框架提供的标识一个http访问是get还是post或者是其他，并且会将path与get中的字符串连接成一个url]
- 上述代码可以从浏览器访问：http://localhost/。
- 下述代码可以从浏览器访问：http://localhost/hello/world [注意path与get中的参数]。

```java
/**
 * @author 54chen(陈臻) [chenzhen@xiaomi.com czhttp@gmail.com]
 * @since 2012-4-10 上午11:14:46
 */
package com.chen.controllers;

import net.paoding.rose.web.annotation.Path;
import net.paoding.rose.web.annotation.rest.Get;

@Path("/hello/")
public class HelloController {

    @Get("world")
    public String index() {
        return "@hello world";
    }
}
```

EOF

- 动态更新版本地址在：https://github.com/XiaoMi/rose/tree/master/chapter_2
- 文中所提及的代码在：https://github.com/XiaoMi/rose/rose-example

rose手册第三章：框架功能参考

3.1 controller层：url对照规则与返回结果规则

3.1.1) url对照规则——最简单的例子

先看看怎样把url和某个方法对应起来。为了方便说明，现在我们来一起完成一个极简版的贴吧。

1)贴吧需要什么功能？

贴吧中当然会有很多“主帖”（topic），“主帖”下会有很多“跟帖”（comment）。
一般，贴吧中最基本的，会有下面这几个功能需要我们完成：

- 显示主帖列表
- 显示单个主帖和它的跟贴
- 显示单个跟贴
- 创建一个主帖
- 创建一个跟贴

2)设计 web API

然后让我们来规划一个REST风格的 web API：（“GET”和“POST”是指HTTP1.1中的请求方法）

- 显示主帖列表
  - GET http://github.com/myforum/topic
- 显示单个主帖和它的跟贴
  - GET http://github.com/myforum/topic/123
- 显示单个跟贴
  - GET http://github.com/myforum/topic/123/comment/456
- 创建一个主帖
  - POST http://github.com/myforum/topic
- 创建一个跟贴
  - POST http://github.com/myforum/topic/123/comment

可以发现一个共同点，所有API中，URI部分的第一级都是“/myforum”（但这并不是规定，仅仅为了演示）。

3)实现 web API

首先新建一个类，这个类的类名必须以“Controller”结尾：

```
@Path("myforum")
public class ForumController {
}
```

注意标注在类(class)上的注解“@Path("myforum")”，这意味着，这个类中定义的所有API的URI，都必须以“myforum”开头，比如“/myforum/xxx”和“/myforum/yyy”等（但“myforum”不一定是整个URI的第一级，比如“/aaa/myforum/bbb”）。

接着，实现第一个API——“GET http://github.com/myforum/topic”：

```
@Path("myforum")
public class ForumController {
    @Get("topic")
    public String getTopics() {
        //显示主帖列表
       return "topiclist";
    }
}
```

因为是“GET”方法，所以在该方法上标注“@Get("")”，URI“/myforum/topic”中的“myforum”已经在“@Path("myforum")”中定义过了，所以只剩下“topic”，于是写“@Get("topic")”。

再看第二个API——“GET http://github.com/myforum/topic/123”。
跟前一个的唯一区别是，后面多了个“/123”，表示主帖id，而这个id当然不是固定的，只有用户点击链接发来请求时才能知道，肿么办？
没关系，rose支持正则表达式！可以这么写：

```
@Get("topic/{topicId:[0-9]+}")
public String showTopic(@Param("topicId") int topicId) {
    //显示单个主帖和它的跟贴
   return "topic";
}
```

与前一个API相比，多了段“/{topicId:[0-9]+}”。正则表达式被大括号"{}"包围，格式为“{ paramName : regularExpression }”，只有请求的URI能被正则表达式匹配时，才会执行这个方法，而被匹配的值将被保存在名为“topicId”的参数中。

同理，实现第三个API，稍微复杂一点：

```
@Get("topic/{topicId:[0-9]+}/comment/{commentId:[0-9]+}")
public String showComment(@Param("topicId") int topicId, @Param("commentId") int commentId) {
    //显示单个跟贴
    return "comment";
}
```

最后两个API使用POST方法，其他与前面相同：

```
@Post("topic")
public String createTopic(){
    //创建一个主帖
    return "topic";
}
@Post("topic/{topicId:[0-9]+}/comment")
public String createComment(@Param("topicId") int topicId){
    //创建一个跟贴
    return "comment";
}
```

完整的代码如下（省略了import语句）：

```
@Path("myforum")
public class ForumController {

    @Get("topic")
    public String getTopics() {
        //显示主帖列表
       return "topiclist";
    }

    @Get("topic/{topicId:[0-9]+}")
    public String showTopic(@Param("topicId") int topicId) {
        //显示单个主帖和它的跟贴
       return "topic";
    }

    @Get("topic/{topicId:[0-9]+}/comment/{commentId:[0-9]+}")
    public String showComment(@Param("topicId") int topicId, @Param("commentId") int commentId) {
        //显示单个跟贴
       return "comment";
    }

    @Post("topic")
    public String createTopic(){
        //创建一个主帖
       return "topic";
    }

    @Post("topic/{topicId:[0-9]+}/comment")
    public String createComment(@Param("topicId") int topicId){
        //创建一个跟贴
       return "comment";
    }
}
```

至此，一个贴吧功能的Controller就编写完成了。

4) 更多细节

除了上面例子中的做法（@Path("")，@Get("")和@Post("")），还可以通过包路径来规划URI。

比如前面例子中的Controller，在API不变的前提下，还可以这么做：

1.在controllers路径下新建一个叫做“myforum”的文件夹。
2.将ForumController从“xxx.controllers”移动到“xxx.controllers.myforum”
并改成下面这样：

```
@Path("")
public class ForumController {
    @Get("topic")
    public String getTopics() {
        //显示主帖列表
       return "topiclist";
    }
    ... ...
}
```

只是将“@Path("myforum")”改成了“@Path("")”。这样做的好处是可以让项目中的代码组织清晰。

3.1.2) 返回结果规则

1) 渲染页面并返回

web开发中最常规的做法是，运行Servlet中的方法，最后将渲染好的页面内容返回。下面说说rose是怎么做的。
上面的贴吧例子中，每个方法的返回值都是一个普通字符串，比如“comment”，意思是，找到web项目中“webapp/views”路径下名叫“comment”的视图文件，比如“comment.jsp”，用这个视图文件来渲染网页结果并返回。
comment.jsp的代码如下：

```
...
<body>
    昵称：${name}<br>
    回复内容：${commentContent}
</body>
...
```

页面中有两个变量——name和commentContent，变量的值是在java代码中设置的，如下：

```
@Get("topic/{topicId:[0-9]+}/comment/{commentId:[0-9]+}")
public String showComment(Model model, @Param("topicId") int topicId, @Param("commentId") int commentId) {
    //显示单个跟贴
    model.add("name", "郭德纲");
    model.add("commentContent", "今天来人不少，我很欣慰啊！");
    return "comment";
}
```

总结一句话，通过rose提供类net.paoding.rose.web.var.Model来设置变量名和变量值，然后在视图文件中用“${paramName}”的方式得到变量值。
变量的值可以是String，boolean，数字，数组，对象(JavaBean)。

如果是对象，使用方法如下：

```
javaBean：
public class Bean{
    private String beanValue;
    public String getBeanValue(){...}
    public String setBeanValue(String beanValue){...}
}
==================
controller中的方法：
@Get("test")
public String test(Model model) {
    Bean bean = new Bean();
    bean.setBeanValue("this_is_a_bean");
    model.add("mybean", bean);
    return "test";
}
==================
test.jsp文件：
...
<body>
    bean里的值：${mybean.beanValue}
</body>
...
==================
输出为：
bean里的值：this_is_a_bean
```

如果是个数组，可以结合JSTL对数组循环访问：

```
controller中的方法：
@Get("test")
public String test(Model model) {
    String[] array = {"111","222","333"};
    model.add("array", array);
    return "test";
}
==================
test.jsp文件：
...
<body>
<c:forEach var="item" items="${array}" varStatus="status">
    打印：${item}<br>
</c:forEach>
```

```
</body>
...
==================
```
输出为：
```
111
222
333
```

2) 还有几种规则？

rose中，controller方法的返回值有下面几种规则：

1.返回普通字符串，如上所述，最常用的做法，渲染视图文件并返回。
2.以“@”开头的字符串，比如“return "@HelloWorld";”，会将“@”后面的字符串“HelloWorld”作为结果返回；
3.以“@json:”开头的字符串，比如:

```
@Get("json")
public String returnJson(){
    JSONObject jo = new JSONObject();
    return "@json:"+jo.toString();
}
```

将会返回一个字符串（jo.toString()），并自动将“HttpServletResponse”中的“contentType”设置为“application/json”。

4.【不推荐使用】以“r:”开头的字符串，比如“return "r:/aaa";”，等效于调用“javax.servlet.http.HttpServletResponse.sendRedirect("/aaa")”，将执行301跳转。
5.【不推荐使用】以“a:”开头的字符串，比如“return "a:/bbb";”，将会携带参数再次匹配roseTree，找到controller中某个方法并执行，相当于“javax.servlet.RequestDispatcher.forward(request, response)”。

3.1.3) 原理

Rose 是一个基于Servlet规范、Spring“规范”的WEB开发框架。

Rose 框架通过在web.xml配置过滤器拦截并处理匹配的web请求，如果一个请求应该由在Rose框架下的类来处理， 该请求将在Rose调用中完成对客户端响应. 如果一个请求在Rose中没有找到合适的类来为他服务，Rose将把该请求移交给web容器的其他组件来处理。

Rose使用过滤器而非Servlet来接收web请求，这有它的合理性以及好处。

Servlet规范以“边走边看”的方式来处理请求， 当服务器接收到一个web请求时，并没有要求在web.xml必须有相应的Servlet组件时才能处理，web请求被一系列Filter过滤时， Filter可以拿到相应的Request和Response对象 ，当Filter认为自己已经能够完成整个处理，它将不再调用chain.doNext()来使链中下个组件(Filter、Servlet、JSP)进行处理。

使用过滤器的好处是，Rose可以很好地和其他web框架兼容。这在改造遗留系统、对各种uri的支持具有天然优越性。正是使用过滤器，Rose不再要求请求地址具有特殊的后缀。

为了更好地理解，可以把Rose看成这样一种特殊的Servlet：它能够优先处理认定的事情，如无法处理再交给其它Filter、Servlet或JSP来处理。这个刚好是普通Servlet无法做到的 ： 如果一个请求以后缀名配置给他处理时候 ，一旦该Servlet处理不了，Servlet规范没有提供机制使得可以由配置在web.xml的其他正常组件处理 (除404，500等错误处理组件之外)。

一个web.xml中可能具有不只一个的Filter，Filter的先后顺序对系统具有重要影响，特别的，Rose自己的过滤器的配置顺序更是需要讲究 。如果一个请求在被Rose处理前，还应该被其它一些过滤器过滤，请把这些过滤器的mapping配置在Rose过滤器之前。

像前面提到过的，RoseFilter的配置，建议按以下配置即可：

```
       <filter>
                <filter-name>roseFilter</filter-name>
                <filter-class>net.paoding.rose.RoseFilter</filter-class>
        </filter>
        <filter-mapping>
                <filter-name>roseFilter</filter-name>
                <url-pattern>/*</url-pattern>
                <dispatcher>REQUEST</dispatcher>
                <dispatcher>FORWARD</dispatcher>
                <dispatcher>INCLUDE</dispatcher>
        </filter-mapping>
```

大多数请况下，filter-mapping 应配置在所有Filter Mapping的最后。 不能将 FORWARD、INCLUDE 的 dispatcher 去掉，否则forward、 include的请求Rose框架将拦截不到。

Rose框架内部采用"匹配->执行"两阶段逻辑。Rose内部结构具有一个匹配树， 这个数据结构可以快速判断一个请求是否应该由Rose处理并进行， 没有找到匹配的请求交给过滤器的下一个组件处理。匹配成功的请求将进入”执行“阶段。 执行阶段需要经过6个步骤处理：“参数解析 -〉 验证器 -〉 拦截器 -〉 控制器 -〉 视图渲染 -〉渲染后"的处理链。

匹配树: 匹配树是一个多叉树，下面是一个例子：

```
ROOT
```

```
    GET="HomeController#index" package="com.xiaonei.xxx.controllers"

/about

    GET="HomeController#about" package="com.xiaonei.xxx.controllers"

/book

    GET="BookController#list" package="com.xiaonei.xxx.controllers.sub"

    POST="BookController#add" package="com.xiaonei.xxx.controllers.sub"

/book/

    /book/{id}

        GET="BookController#show" package="com.xiaonei.xxx.controllers.sub"

/help

    GET="HomeController#help" package="com.xiaonei.xxx.controllers"
```

ROOT代表这是一个根地址，也就是 http://localhost/ 代表的地址；

ROOT的下级有个GET结点，代表对该地址支持GET访问，不支持POST等其它访问，如果进行POST访问将以405错误回应。

/book代表这是一个/book地址，也就是 http://localhost/book 代表的地址；

/book下级有GET、POST两个结点，说明它支持GET和POST方法，根据HTTP语义，GET代表浏览，POST代表追加(向一个集合中追加一个条目)。

/book下还有/book/地址，这个地址有点特别，它以'/'结尾，但实际它不会被任何地址访问到，rose对http://localhost/book/的处理会将它等价于 http://localhost/book 。

这个特别的地址的存在完全是匹配树结构所需导致的，但不对实际匹配有任何坏的影响，所以也没有任何GET、POST等子结点。

/book/{id}代表是一个/book/123456、/book/654321这样的地址，当然这可以支持正则表达式的。

大部分情况下，匹配树的结构和实际的URI结构会一致，也因此匹配树的深度并不固定，每一个中间结点或叶子节点都有可能代表一个最终的URI地址，可以处理GET、POST等请求。对于那些匹配树存在的地址，但没有GET、POST、DELETE等子结点的，一旦用户请求了该地址，rose将直接把该请求转交给web容器处理，如果容器也不能处理它，最终用户将得到404响应。

匹配过程: Rose以请求的地址作为处理输入(不包含Query串，即问号后的字符串)。如果匹配树中存在对应的地址，且含有对应请求方法(GET、POST、PUT、DELETE)的，则表示匹配成功；如果含有其他方法的，但没有当前方法的（比如只支持GET，但当前是POST的），则也表示匹配成功，但最后会以405响应出去；如果所给的地址没有任何支持的方法或者没有找到匹配地址的，则表示匹配失败。1.0.1不支持回朔算法，1.0.2将支持部分回朔算法(待发布时再做详细介绍)。

参数解析: 在调用验证器、拦截器 控制器之前，Rose完成2个解析：解析匹配树上动态的参数出实际值，解析控制器方法中参数实际的值。参数可能会解析失败(例如转化异常等等)，此时该参数以默认值进行代替，同时Rose解析失败和异常记录起来放到专门的类中，继续下一个过程而不打断执行。

3.2 controller层：拦截器支持

3.2.1 拦截器作用

- 面向切面编程（AOP）方法可以让一个项目更加关注核心逻辑，常见的一些最佳实践包括
  - 权限
  - 缓存
  - 错误处理
  - 延时加载
  - 调试
  - 持久化
  - 资源池
  - 等等。。。
- 而此处的拦截器目标是在controller层提供各种在controller执行前、执行后的代码切入，以达到各种可AOP的目标。
- 简单地说，拦截器能干的事情就是当你的项目写了一半时发现缺少啥全局要做的事情（比如需要验证权限），不用担心，搞一个拦截器就是了。

3.2.2 拦截器例子

```
public class AccessTrackInterceptor extends ControllerInterceptorAdapter {
    public AccessTrackInterceptor() {
    setPriority(29600);
    }
    @Override
    public Class<? extends Annotation> getRequiredAnnotationClass() {
```

```
        return PriCheckRequired.class; // 这是一个注解，只有标过的controller才会接受这个拦截器的洗礼。
    }
    @Override
    public Object before(Invocation inv) throws Exception {
        // TODO ....
    return super.before(inv);
    }

    @Override
    public void afterCompletion(final Invocation inv, Throwable ex) throws Exception {
    // TODO ....
    }
}
```

需要注意几点：

- 拦截器要放在controllers下(高级用法:打在rose-jar包里，参见5.1)
- 继承net.paoding.rose.web.ControllerInterceptorAdapter
- 按照实现的方法名，在controller执行前、中、后执行：
  - before：在controller执行前执行。
  - after：在controller执行中（后）执行，如果一个返回抛出了异常，则不会进来。
  - afterCompletion：在controller执行后执行，不论是否异常，都会进来。
  - isForAction：定义满足某条件的才会被拦截。

3.2.3 拦截器可动的位置细节

- 上面都讲得差不多了，实际上还有不少地方可以拦截的：
  - isForDispatcher：根据响应的情况判断是否拦截，比如说是正常请求、内部forward、还是include （但是没用过）
  - setPriority：设置一个数字表示拦截优先级，当有多个拦截器时，要精准控制，数字小的内层，大的在外层，在最外层的before方法最先执行，大家都执行完后它的after才最后执行。
  - round：这才是真正的controller执行中执行，不过用得很少。
  - getRequiredAnnotationClass：返回一个Annotation class name，表示这个拦截器只对此Annotation标过的controller才生效。常用。

3.2.4 实际应用场景

- 全站是否登录判断相关的逻辑，写在一个拦截器里，一次完成后，其他地方不再关心这个代码，在需要登录才能做的controller上注解一下，表示需要被执行拦截。
- 日志收集的逻辑，在一个拦截器里进行当前的access log记录。
- 权限体系的逻辑，写在一个拦截器里，在对应的操作上作注解，拦截器中进行细节的判断，新加的api也只是需要一次注解就得到了权限的判断。

[文中所提代码均在 https://github.com/XiaoMi/rose/tree/master/rose-example]

3.3 controller层：ErrorHandler支持

3.3.1 ErrorHandler的作用

- 一般来说传统的编程都会到处去try，特别是java里，try来try去的（如果你用erlang一定就知道，已经知道的可能性，怎么能叫异常？都try了还是让它崩了算了。。。）。
- 如果打开你的项目，每个java文件中的代码都有一堆的try，那这时候就是ErrorHandle上阵的时候了。
- ErrorHanle致力于：统一捕捉和处理各种异常，可区分对待和返回；统一的出错体验。
- 非常类似做web开发时的500统一出错页面这样的东东。

3.3.2 示例

```
/**
 * @author chenzhen@xiaomi.com
 * 2010-12-1
 */

package com.chen.controllers;

import net.paoding.rose.web.ControllerErrorHandler;
import net.paoding.rose.web.Invocation;

public class ErrorHandler implements ControllerErrorHandler {

    public Object onError(Invocation inv, Throwable ex) throws Throwable {

        // TODO logger.error("handle err:", ex);

        return "@error";
```

```
    }
}
```

这是这么简单，不用怀疑！

3.3.3 放在哪里才能生效？

- 放在controllers目录下，和controller们在一起（幸福快乐地生活）。
- 一般来讲，ErrorHandler都是用在web项目里，在最快层起作用。
- 所有的方法都可以尽情地向处throws Exception了。
- 不需要再try了。

```
@Path("")
public class HelloController {
    @Get("")
    public String index2() throws Exception {
        return "@hello world";
    }
}
```

3.3.3 有用的例子: 不同的异常类型做不用的事情

```
/**
 * @author chenzhen@xiaomi.com
 * 2010-12-1
 */

package com.chen.controllers;

import net.paoding.rose.web.ControllerErrorHandler;
import net.paoding.rose.web.Invocation;

public class ErrorHandler implements ControllerErrorHandler {

    public Object onError(Invocation inv, Throwable ex) throws Throwable {

        // TODO logger.error("handle err:", ex);
        if (ex instanceof RuntimeException) {
            return "@runtime";
        }
        return "@error";
    }
}
```

文中所提及代码均在 https://github.com/XiaoMi/rose/tree/master/rose-example 提供。

3.4 controller层：自定义http参数支持

3.4.1 http参数支持的一些前言

- 我们把一个controller的类里的一个方法叫做action，它实际对应用户看到的一个url。
- 在action里可以接收各种各样的参数，也可以自己定义需要的参数。
- rose自己定义了一些常见的类型，基本上很有机会会用到自己定义，但是在某些情况下，也是个不错的选择：
    - 用来对指定的参数类型的值进行固定的修改和赋值。

3.4.2 看一个例子

ChenBeanResolver.java放在controllers目录下：

```
public class ChenBeanResolver implements ParamResolver {

    @Override
    public Object resolve(Invocation inv, ParamMetaData metaData) throws Exception {
        for (String paramName : metaData.getParamNames()) {
            if (paramName != null) {
                Chen chen = new Chen();
                String value1 = inv.getParameter("chen1");
```

```
                String value2 = inv.getParameter("chen2");
                chen.setChen1(value1);
                chen.setChen2(value2);
                return chen;
            }
        }
        return null;

    }

    @Override
    public boolean supports(ParamMetaData metaData) {
        return Chen.class == metaData.getParamType();
    }

}
```

- 上述代码的意思：
  - 如果在action里一个参数的类型是Chen(com.chen.model.Chen)，就会走这个resolver，这里对两个参数进行了组装。
  - 用户如果访问的参数里传入了chen1和chen2的值，则会直接组装出来一个Chen对象。
- 配合上述resolver的action代码为：

```
@Get("/param")
    public String param(Chen chen) throws Exception {
        return "@hello world:" + chen.getChen1() + ":" + chen.getChen2();
    }
```

- 用户访问：http://127.0.0.1/param?chen1=1&chen2=2 将会返回： ** hello world1️⃣2

3.4.2 rose内置的参数支持

除了上述的自定义resolver外，rose还内置了丰富的resolver，都是大家的经验总结，使用起来会非常方便，它们是：

- 所有的基础java类型，都可以直接使用，rose进行自动转换，比如在action中的类型为long id，则id可以转为数字，不再需要从string转为long。
- array/map/bean同样可用，它们的接收参数规则为：
  - ?id=1,2,3,4 或者 ?id=1&id=2&id=3 对应 @Param("id") int[] idArray
  - ?map:1=paoding&map:2=rose 对应 @Param("map") Map map
  - POST http://127.0.0.1/user?id=1&name=rose&level.id=3 对应接收代码：

```
    @Post
    public String post(User user) {
         return "@" + user.getId() + "; level.id=" + user.getLevel().getId();
    }
```

- 代码中User是一个自定义的bean，有属性id,name,level等。

文中所提及代码均在 https://github.com/XiaoMi/rose/tree/master/rose-example 提供。

3.5 controller层：统一的参数验证办法

3.5.1 用来做什么

- 我们把的参数验证办法叫ParamValidator
- 一般来说，像比如说验证http传来的参数是不是为空呀啥的（发挥你的想象力）。
- 好处在于不用再重复地写if else

3.5.2 怎么用

- 来看一个例子，验证用户的参数不可为空(灰常灰常的实用)：

```
public class NotBlankParamValidator implements ParamValidator {

    @Override
    public boolean supports(ParamMetaData metaData) {
        return metaData.getAnnotation(NotBlank.class) != null;
    }

    @Override
    public Object validate(ParamMetaData metaData, Invocation inv, Object target, Errors errors) {
        String paramName = metaData.getParamName();
```

```
        String value = inv.getParameter(paramName);
        if (StringUtils.isBlank(value)) {
            return "@参数不能为空";
        }
        return null;
    }
}
```

解读：

- 放到controllers下
- 实现ParamValidator
- 实现supports方法，这个方法用来做判断是否要验证当前得到的http参数，一般都用个注解来判断比较文艺
- 实现validate方法，这里是主要逻辑
    - metaData里放的是参数的原型
    - inv是rose的基础调用
    - target是这个参数的最后解析结果，参看上一节里提到的东西
    - errors是这个参数解析时出来的错误
- NotBlank是一个自己定义的annotation

3.5.3 使用时action长什么样？

- 下面的代码是action中使用时长的样子:

```
@Get("/notBlank")
public String notBlank(@NotBlank @Param("messages") String messages) throws Exception {
    return "@hello world";
}
```

解读：

- 当遇到NotBlank注解的参数时，会自动执行参数判断
- 如果messages为空，则会得到“参数不能为空”的返回

文中所提及代码均在 https://github.com/XiaoMi/rose/tree/master/rose-example 提供。

3.6 controller层：一闪而过的信息，flash支持

3.6.1 需求描述

- 历史上，做web的需求时，经常遇到一个情况：在A页面修改/添加/删除了信息，提交，提示“修改/添加/删除成功！”。
- rose的flash（并非你所想象的adobe的flash）建设性地使这一需求在开发过程中简单快捷化。

3.6.2 使用过程

- 使用过程会很愉快，在两个action之间，通过return "r:/xxx"来跳转（实际是301），只需要在第一个action里使用flash.put，在第二个action里使用flash.get即可。

```
@Get("/flash1")
public String flashStep1(Flash flash) {
    flash.add("msg", "修改成功！");
    return "r:/flash2";
}

@Get("/flash2")
public String flashStep2(Invocation inv, Flash flash) {
    inv.addModel("info", flash.get("msg"));
    return "flash";
}
```

- 上述两个action中，当访问flash1时，一句flash信息被写入，快速跳转到flash2的地址。
- flash2地址中接收到这个flash信息后写到model中。
- 还需要在flash2的模板里去显示这个变量。

```
<%@ page contentType="text/html;charset=UTF-8"%>
提示信息：${info}
```

3.6.3 注意事项

- flash功能利用了浏览器的cookies功能，如果用户的环境不能使用cookies将不会有任何效果。

文中所提及代码均在 https://github.com/XiaoMi/rose/tree/master/rose-example 提供。

3.7 controller层：门户必备portal支持

3.7.1 什么是portal?

*字面意思，做门户用的。 *简单来说，把一个网页分成了N个区域，每个区域由不同的action去执行，多线程并行提高cpu使用率。

3.7.2 使用例子

*要使用portal，必须先在web.xml里声明所使用的线程池大小：

```
    <context-param>
        <param-name>portalExecutorCorePoolSize</param-name>
        <param-value>1024</param-value>
     </context-param>
```

*然后看示例代码：

```
    @Get("/3.7")
     public String portal(Portal portal) {
         portal.addWindow("p1", "/wp1");
         portal.addWindow("p2", "/wp2");
         return "portal";
     }

     @Get("/wp1")
     public String portal1() {
         return "@this is p1";
     }

     @Get("/wp2")
     public String portal2() {
         return "@this is p2";
     }
```

*然后在第一个action中的portal.jsp中写到：

```
<%@ page contentType="text/html;charset=UTF-8"%>
portal演示信息：
<br>
${p1}
<br>
${p2}
```

*当我们部属好了之后，访问http://127.0.0.1/3.7 *将从浏览器中得到： *portal演示信息： *this is p1 *this is p2

3.7.3 这样子做的好处

*更加充分地使用多核cpu。 *更加方便多人协作时对项目进行模块划分，搞的时候，按照url一分，一个url一个模块，所有的页面都可以切成小的豆腐块，所以，你懂的。

3.7.4 过去的经典事迹

- 2010年的6月9日晚上7点"圣战"
- http://www.54chen.com/architecture/rose-open-source-portal-framework.html

文中所提及代码均在 https://github.com/XiaoMi/rose/tree/master/rose-example 提供。

3.8 controller层：门户必备pipe支持

3.8.1 什么是pipe?

- pipe起源于facebook的工程师对他们网页提速的方案：将网页分解为Pagelets的小块（在rose叫做window的小块），然后通过后端多重管道运行，以达到性能的最佳。

- pipe巧妙使用了http 1.1连接有timeout的机制，充分使用一次http连接来传递数据。
- pipe可使用户在大多数浏览器中感受到延迟减少了一半。

3.8.2 与facebook的bigpipe相比rose pipe如何？

- fb并未在开源项目中公布过使用方法
- bigpipe神似是php+js搞定的
- rose pipe可以自由选择线程池大小，完全出自上一节的portal的基础
- 完全实现bigpipe功能，天然的对业务开发者透明

3.8.3 看实例

HelloController.java

```
    @Get("/3.8")
    public String pipe(Pipe pipe) {
        pipe.addWindow("p1", "/wp1");
        pipe.addWindow("p2", "/wp2");
        return "pipe";
    }
```

- 长得是不是很像上一节里提供的action？
- 不同在于jsp文件中：

```
<%@ page contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://paoding.net/rose/pipe" prefix="rosepipe"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>portal/pipe演示信息</title>
<script type='text/javascript' src='/js/rosepipe.js'></script>
</head>
<body>

portal/pipe演示信息：
<br>
<div id="p1"></div>
<br>
<div id="p2"></div>

</body>
</html>
<rosepipe:write>${p1}</rosepipe:write>
<rosepipe:write>${p2}</rosepipe:write>
```

- 当使用jsp文件时，需要在尾部使用rosepipe:write标签
- 如果是使用vm文件，可以不写这个标签

3.8.4 总结

- 上述代码中p1 p2两个window会同时在多个线程中执行，如果是portal，那会多个线程执行完成一起返回，而pipe则会用js反写的方式，一个线程一个线程地返回给用户。
- pipe是个好物件
- 使用时jsp一定不要忘记尾部的标签
- 使用时web.xml一定不要忘记声明使用的线程池大小
- 久经考验

  文中所提及代码均在 https://github.com/XiaoMi/rose/tree/master/rose-example 提供。

3.9 controller层：上传文件

3.9.1 其实很简单

- 添加依赖包:commons-io.jar
- html中使用 enctype="multipart/form-data",method="POST"
- 直接看后端代码吧。

```
    @Post("/doUpload")
    public String doUpload(@Param("file") MultipartFile file) {
        return "@ upload ok!" + file.getOriginalFilename();
    }
```

3.9.2 其他

- 可以同时接收所有的文件

```
// 不声明@Param
// files可以是一个数组或者List
public String upload(MultipartFile[] files) {
    return "@ok-" + Arrays.toString(files);
}
```

- 同时也可以使用@Param传递不同的name。

- 文中所提及代码均在 https://github.com/XiaoMi/rose/tree/master/rose-example 提供。

3.A DAO层：DAO的基本配置与使用

本章开始进入对DB层的支持，同进也是日常开发用得最多的章节。

3.A.1 什么是jade?

- jade大概是java access data layer的意思吧，具体的来由，在章节写到末尾的时候，我再找qieqie和liaohan大侠们写一写编年史。
- 用jade的好处在于，尽可能减少重复的从db把数据对bean进行装配的过程，统一入口，隔离业务逻辑，方便review。
- jade是在spring完成的数据层的良好实践总结，无缝接入rose中，可以算得上是rose亲密无间的好模块。

3.A.2 引入基础配置

- 要开始使用jade，一定要先引用jade的基础包：

pom.xml

```
        <dependency>
            <groupId>com.54chen</groupId>
            <artifactId>paoding-rose-jade</artifactId>
            <version>1.1</version>
        </dependency>
```

- 除了需要jade的包外，还需要引入数据源连接池的jar，这里使用了dbcp，当然了mysql-connector也是必不可少的，还是在pom.xml中添加：

```
       <dependency>
            <groupId>commons-dbcp</groupId>
            <artifactId>commons-dbcp</artifactId>
                        <version>1.2.2</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.10</version>
        </dependency>
```

- 在pom中引入了依赖之后，需要定义一个数据源，这里先不考虑多个数据源的情况。

在war项目的applicationContext.xml中增加数据源定义：

```
    <!-- 数据源配置 dbcp -->
    <bean id="jade.dataSource.com.chen.dao" class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
        <property name="url"
            value="jdbc:mysql://127.0.0.1:3306/test?useUnicode=true&amp;characterEncoding=utf-8"></property>
        <property name="username" value="test"></property>
        <property name="password" value="test"></property>
        <!-- 运行判断连接超时任务的时间间隔，单位为毫秒，默认为-1，即不执行任务。 -->
        <property name="timeBetweenEvictionRunsMillis" value="3600000"></property>
```

```
        <!-- 连接的超时时间，默认为半小时。 -->
        <property name="minEvictableIdleTimeMillis" value="3600000"></property>
    </bean>
```

- 这里假设了mysql已经安装在本地了，用户名为test，密码为test。
- jade约定了bean的id为jade.dataSource.classPackageName。
- jade约定了这个bean的有效范围为classPackageName所有的DAO。
- jade约定了除非有专门的定义，所有的子目录也受bean上的classpackageName所影响。

3.A.3 第一个读取数据库的实例

- 先需要准备一下数据库：

```
create table test (int id, varchar(200) msg);
insert into test values(111,'adfafafasdf');
```

- 然后准备简练的DAO声明：

```
@DAO
public interface TestDAO {
    @SQL("select id,msg from test limit 1")
    public Test getTest();
}
```

- Test是一个class，里面有标准的getter和setter。

- 然后从一个类去调用它：

```
@Service
public class TestService {

    @Autowired
    private TestDAO testDAO;

    public Test getTest() {
        return testDAO.getTest();
    }
}
```

- 当然也可以直接用controller就调用它显示了，当然这不是一个大型项目良好的架构，架构问题在最后的章节里讲述。

3.A.4 查看演示代码

- rose-example项目的HelloController已经准备好了一个action做为此节的总结。
- 你可以将此节发布到resin或者tomcat后访问：http://127.0.0.1/3.10 查看结果。

- 文中所提及代码均在 https://github.com/XiaoMi/rose/tree/master/rose-example 提供。

3.B DAO层：DAO进阶：SQLParm支持和表达式SQL

3.B.1 SQLParam介绍：DAO方法传递参数

- SQLParam作为DAO支持中的参数传递使者，可以传递一个常见的变量，也可以是一个自定义的对象。
- 比如：

```
    @SQL("insert into test (id,msg) values (:t.id,:t.msg)")
    public void insertTest(@SQLParam("t") Test test);
```

- 上列中Test对象通过t传递到sql执行中去，并且可以分别使用其中的属性。这感觉是不是很自然？

- 当然，如果是一个int、long、String等自在不言中。

- 当是list时，会有自动的batch操作，将sql拆为多条sql执行。这个小技巧会在后面的章节里讲。平时很少用到。

3.B.2 ReturnGeneratedKeys介绍：返回刚刚插入的ID号

- 特别是使用mysql开发的广大劳苦大众，常常会使用到auto_increament的字段。
- 当一条insert语句在执行的时候，我们常常会去需要拿它的当前的自增id是多少。

```
@ReturnGeneratedKeys
 @SQL("insert into test (id,msg) values (:t.id,:t.msg)")
 public int insertTest(@SQLParam("t") Test test);
```

- 如上述代码所示，只需要加上一个@ReturnGeneratedKeys即可返回当前的id

3.B.2 表达式的支持

- 多变的业务需求决定了我们的sql是复杂的，需要有条件地执行。
- 如果每种条件都去写DAO中的SQL，那DAO的变得很大。
- 常常会有动态产生sql的需求。
- jade支持一些常规的表达式。

- 语法一：常见的变量赋值
    - 冒号（:）表示这是一个变量，比如上面的例子里的 :t.id，它会被一个值替换。
- 语法二：字符串连接
    - 连续的井号（##） 表示后面的变量作字符串连接
    - 如下例中的partition变量，还请不要误解，分表不是这样做的，下一章会介绍标准的分表设置。

```
@SQL("SELECT user_id, device_token FROM test_##(:partition) LIMIT :limit")
 public List<Test> getTests(@SQLParam("partition") int partition, @SQLParam("limit") int limit);
```

- 语法三：条件选择
    - 井号if（#if{}）用于表示当条件满足时sql拼接。

```
@SQL("SELECT user_id, device_token FROM test_##(:partition) #if(:user>0){ where user_id=:user } LIMIT :limit")
 public List<Test> getTestsIf(@SQLParam("partition") int partition, @SQLParam("limit") int limit, @SQLParam("user") int
```

- 其他语法：还有for循环，实际使用少。
- 典型地，一般的select in查询，可以直接传入list，例如下例中的ids变量：

```
@SQL("SELECT user_id, device_token FROM test_##(:partition) where user_id in(:ids)")
 public List<Test> getTestsByIds(@SQLParam("partition") int partition, @SQLParam("ids") List<Integer> ids);
```

- 文中所提及代码均在 https://github.com/XiaoMi/rose/tree/master/rose-example 提供。

3.C DAO层：分表设置

欢迎顺利进入本章，如果您的企业需要这一节的内容，那么说明用户量很有前途，如果使用了本节的内容，不防向czhttp@gmail.com发信一封以表谢意，我们会很高兴收到各种反馈。

3.C.1 mysql分表的常规做法

以下是个人从业经验中的分表规则：

- 按照 id % 100 分为一百份
- 按照 id % 16 分为十六份
- 按照 id/10 % 10 分为十份
- 按照 id%10 分为十份

以上分表规则特别在mysql中使用机会比较多，各有优势，没有对错，只有最好与最不好用。

3.C.2 使用分表第一步：添加新的依赖

要使用分表，需要添加新的依赖，由bmw提供的bmwutils。

```
<dependency>
    <groupId>com.54chen</groupId>
    <artifactId>bmwutils</artifactId>
    <version>0.0.2</version>
</dependency>
```

3.C.3 使用分表第二步：设置applicationContext.xml分表规则

在开写代码之前，需要告诉DAO是哪个表需要分表，按照什么规则分，分多少份。

```
<!-- 以下配置为分表设置 -->
<bean id="jade.routerInterpreter" class="com.xiaomi.common.service.dal.routing.RewriteSQLInterpreter">
        <property name="routingConfigurator" ref="jade.routingConfigurator" />
</bean>
<bean id="jade.routingConfigurator" class="com.xiaomi.common.service.dal.routing.RoutingConfigurator">
        <property name="partitions">
              <list>
                    <value>hash:test:id:test_{0}:100</value>
              </list>
        </property>
</bean>
```

- 此处配置中，partitions参数为一个list，可以对多个table进行定义。
    - hash:test:id:test_{0}:100 表示：使用hash这种办法，将test这个表，按照id的值，分成100份，每份的表名为test_x

3.C.4 使用分表第三步：bmwutils支持的分表办法

- (hash)上例中的hash: 最常用的 id % 100 就是这种办法。该办法会把传入的值先进行转为数字后与定义的份数进行取模（%）。
- (direct)最直接的一种：用的少一些，没有什么规则，直接根据第四个正则，与第三位传入的值进行替换。假设有个人名表，按照字母分表可以用。name_A,name_B,name_C...
- (round)轮循：根据设置，按照调用sql的情况，轮流使用各个表。
- (range)范围：一般用来做日期范围的分表，比如说微博类的，可变值为一个时间，当时间传入时，第三位支持log*{yyyy}* log{yyyy_MM}等时间格式的替换，可轻松做到按周、月、年分表。
- (xm-hash)小米hash：一种古怪的办法，按照传入值的十位进行取模的分表方案。
- (xm-str-hash)小米字符串hash：将字符串按照固定算法变成long之后，再按照小米hash逻辑处理。
- (hex-hash)16进制分表：固定256份以内，传入的值按照16进制转换后按hash求模。

3.C.5 使用分表第四步：写DAO代码@ShardBy

```
@SQL("SELECT user_id, device_token FROM test where user_id =:id")
public List<Test> getTestsById(@ShardBy @SQLParam("id") int id);
```

与不分表的dao相比，只多了一个shardBy，标识按照这个参数值分表。

- 文中所提及代码均在 https://github.com/XiaoMi/rose/tree/master/rose-example 提供。

rose手册第四章: 安全

- 在controller中，如果你不想将一个方法公开对外，那就不要将其标识为public，因为public的方法会被rose认为是一个要公开的action。

rose手册第五章：FAQ 常见问题

5.1 如何打一个可被rose识别的jar包

如果你使用maven，在pom中添加如果定义之后，你打出来的jar包就会被rose扫描到并且引入到上下文环境中：

```
<plugin>
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-jar-plugin</artifactId>
     <configuration>
         <archive>
             <manifestEntries>
                 <Rose>*</Rose>
             </manifestEntries>
         </archive>
     </configuration>
 </plugin>
```

- 用途：
    - 比如说你做了一个拦截器，在多个项目里需要相同的逻辑，那只需要把这个拦截器做在一个jar包里，声明是rose支持的包即可被使用。

5.2 会被认成batch执行的sql返回

如果你的sql在执行update insert delete，并且dao的第一个参数是list类的多个值，那这条sql会被拆成多条sql依次执行，执行的结果会以各条sql的返回组成的数组返回。

5.3 一个良好的大型WEB项目架构实践

我们一般会把项目规定为：controller/service/biz/dao层，不能跨层调用，只在service层允许同时调用子层多个方法。

rose手册第六章：附录

附录里会有qieqie同学亲笔的rose编年史和发展规划。