



Chrome 性能挑战赛

挑战专家观点

赢取Google I/O大会最后参会名额
(含往返机票及住宿一周)



论坛首页 → Java编程和Java企业应用版 → Struts → 关于支持RESTful的思考

全部 Hibernate Spring Struts iBATIS 企业应用 设计模式 DAO 领域模型 OO Tomcat SOA JBoss Swing Java综合

最成熟稳定甘特图控件,支持Java和.Net

浏览 3298 次

锁定老帖子 主题: 关于支持RESTful的思考

该帖已经被评为良好帖

作者

正文

javatar

等级: ★★★★★

性别:

文章: 251

积分: 578

来自: 杭州699号

我现在离线

< >

猎头职位: 北京: ITeye网站诚聘网站产品UED主管

现在基本上所有的MVC框架都叫嚷着支持RESTful(<http://zh.wikipedia.org/wiki/REST>), 最近写的Struts(for)RCP(<http://struts4rcp.googlecode.com>)也来凑下热闹, 这里讲下基本思路, 作个参考。

REST的一些要求, 如:

1. 客户端和服务端结构
2. 连接协议具有无状态性
3. 能够利用Cache机制增进性能
4. 层次化的系统
5. Code On Demand - Javascript

通过RCP/RIA和HTTP协议本身就可以达到, 就不多说了, 主要关注REST的设计风格, 如:

1. 资源是由URI来指定。
2. 对资源的操作包括获取、创建、修改和删除资源, 这些操作正好对应HTTP协议提供的GET、POST、PUT和DELETE方法。
3. 通过操作资源的表形来操作资源。
4. 资源的表现形式则是XML或者HTML, 取决于读者是机器还是人, 消费web服务的客户软件还是web浏览器。当然也可以任何其他其他的格式。

下面一一论述以上四点的实现:

1. URI数据映射

鉴于RESTful要从URI上取值, 并注入到相应属性, 这就需要一种方式, 来声明哪一截数据应该注入到哪个属性, 在参考了多个RESTful框架后, 决定采用常见的"/catalogs/{categoryId}/books/{bookId}"风格声明URI数据提取方式, 如:

Java代码

```
1. @Path("/users/{id}")
2. public class UserManagerAction extends ResourceAction<User> {
3.
4.     // .....
5.
6. }
```

2. 方法分派

关于方法的对应关系, 有很多做法, 如:

(1) 函数名与请求类型保持一致, 如:

Java代码

```
1. @Path("/users/{id}")
```

相关文章:

- 关于webwork字段值自动绑定的困惑
- WebWork2&Struts1.1(1):Forum
- Spring3MVC+MyBatis+ExtJs3整合开发系列之三: 人员管理模块


推荐群组: JBPM @net

更多相关推荐

```
2. public class UserManageAction extends ResourceAction<User> {
3.
4.     @Override
5.     public void post(User user) throws Exception {
6.         // 除了ID以外的值, 通过表单传入, 表单中有ID值无效(会被覆盖掉)
7.         userService.create(user);
8.     }
9.
10.    @Override
11.    public void put(User user) throws Exception {
12.        // 除了ID以外的值, 通过表单传入, 表单中有ID值无效(会被覆盖掉)
13.        userService.update(user);
14.    }
15.
16.    @Override
17.    public void delete(User user) throws Exception {
18.        userService.delete(user.getId());
19.    }
20.
21.    @Override
22.    public User get(User user) throws Exception {
23.        return userService.get(user.getId());
24.    }
25.
26. }
```

但感觉这样, 函数名对业务开发人员不友好。

(2) 取一个更直观的名称与之对应, 如:

Java代码 

```
1. @Path("/users/{id}")
2. public class UserManageAction extends ResourceAction<User> {
3.
4.     @Override
5.     public void create(User user) throws Exception {
6.         // 除了ID以外的值, 通过表单传入, 表单中有ID值无效(会被覆盖掉)
7.         userService.create(user);
8.     }
9.
10.    @Override
11.    public void update(User user) throws Exception {
12.        // 除了ID以外的值, 通过表单传入, 表单中有ID值无效(会被覆盖掉)
13.        userService.update(user);
14.    }
15.
16.    @Override
17.    public void delete(User user) throws Exception {
18.        userService.delete(user.getId());
19.    }
20.
21.    @Override
22.    public User get(User user) throws Exception {
23.        return userService.get(user.getId());
24.    }
25.
26. }
```

上面四个方法分别对应POST,PUT,DELETE,GET四个HTTP请求方法。

(3) 不限制名称, 用标注在函数上进行标识, 如:

Java代码 

```
1. @Path("/users/{id}")
2. public class UserManageAction extends ResourceAction<User> {
3.
```

```
4.         @Post
5.         public void createUser(User user) throws Exception {
6.             // 除了ID以外的值, 通过表单传入, 表单中有ID值无效(会被覆盖掉)
7.             userService.create(user);
8.         }
9.
10.        @Put
11.        public void updateUser(User user) throws Exception {
12.            // 除了ID以外的值, 通过表单传入, 表单中有ID值无效(会被覆盖掉)
13.            userService.update(user);
14.        }
15.
16.        @Delete
17.        public void deleteUser(User user) throws Exception {
18.            userService.delete(user.getId());
19.        }
20.
21.        @Get
22.        public User getUser(User user) throws Exception {
23.            return userService.get(user.getId());
24.        }
25.    }
26. }
```

(4) 配置文件声明, 如:

Java代码 ☆

```
1. @Path("/users/{id}")
2. public class UserManagerAction extends ResourceAction<User> {
3.
4.     public void createUser(User user) throws Exception {
5.         // 除了ID以外的值, 通过表单传入, 表单中有ID值无效(会被覆盖掉)
6.         userService.create(user);
7.     }
8.
9.     public void updateUser(User user) throws Exception {
10.        // 除了ID以外的值, 通过表单传入, 表单中有ID值无效(会被覆盖掉)
11.        userService.update(user);
12.    }
13.
14.    public void deleteUser(User user) throws Exception {
15.        userService.delete(user.getId());
16.    }
17.
18.    public User getUser(User user) throws Exception {
19.        return userService.get(user.getId());
20.    }
21.
22. }
```

Xml代码 ☆

```
1. <bean id="userManagerAction" class="com.xxx.UserManagerAction">
2.     <property name="postMethodName" value="createUser">
3.     <property name="putMethodName" value="updateUser">
4.     <property name="deleteMethodName" value="deleteUser">
5.     <property name="getMethodName" value="getUser">
6. </bean>
```

最终觉得第2种方案比较简单友好, 并且客户端便于创建对等接口, 标注和配置也可以考虑支持。


还有一个问题是, ResourceAction应不应该提供list, find等函数, 用于列表和查找, 如:

Java代码 ☆

```
1. @Path("/users/{id}")
```

```
2. public class UserManageAction extends ResourceAction<User> {
3.
4.     // create(), update(), delete(), get()
5.
6.     @Override
7.     public Collection<User> list() throws Exception {
8.         return userService.findAll();
9.     }
10.
11.    @Override
12.    public Collection<User> find(User user) throws Exception {
13.        return userService.find(user);
14.    }
15. }
```

这些已经不是同一个资源了，而是同一类型的资源，如果分开应该是：


Java代码 

```
1. @Path("/users")
2. public class UserListAction extends ResourceAction<Collection<User>> {
3.
4.     @Override
5.     public Collection<User> get(User user) throws Exception {
6.         return userService.find(user); // 如果条件为空即为全部
7.     }
8.
9. }
```

如果定义在一起，@Path就需要声明多个资源路径，

如：@Path({"users/{id}", "users"})

另一个相似的问题是，ResourceAction应不应该提供add, edit方法，用于跳转到新增页面和编辑页面，如：

Java代码 

```
1. @Path("/users/{id}")
2. public class UserManageAction extends ResourceAction<User> {
3.
4.     // create(), update(), delete(), get()
5.
6.     @Override
7.     public User add() throws Exception {
8.         return new User();
9.     }
10.
11.    @Override
12.    public User edit(User user) throws Exception {
13.        return userService.get(user.getId());
14.    }
15. }
```

如：@Path({"users/{id}", "users", "users/add", "users/{id}/edit"})

当然，可以用命名约定来减少路径的个数，问题在于add和edit已经超出(甚至违背了)RESTful风格，

因为RESTful要求客户端Code On Demand，要求在客户端缓存和管理状态，

像新增页面应该在客户端直接处理，而编辑页面实际上是调用get方法取值填充，只是非RIA的Web应用需要这种间接页面，如果用RESTful风格第4点“资源多重表述”来解决，可能更合理，

edit只是资源“users/{id}”的一种GET展示形式，也就是客户端设置“Accept”信息头为“edit”，

但这样存在一个问题，浏览器不能友好的输入请求头信息。

另外，add是否需要作为特殊值，以避免冲突，因为可能某个用户的ID就是add，当然，如果ID都是数字就没问题。

再者，根据RFC2616(<http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>)的规定：

RFC2616 写道

9.1.2 Idempotent Methods

Methods can also have the property of "idempotence" in that (aside from error or expiration issues) the side-effects of N > 0 identical requests is the same as for a single request. The methods GET, HEAD, PUT and DELETE share this property. Also, the methods OPTIONS and TRACE SHOULD NOT have side effects, and so are inherently idempotent.

要求除POST以外的方法都具有幂等性(idempotence),
也就是调用这些方法n次与调用1次的结果一致,
比如, 你删除一个资源8次, 和删除这个资源1次没什么区别,
还要求OPTIONS和TRACE方法无副作用(side effects),
也就是不修改任何状态(包括数据库数据与对象属性),
调用这些方法n次与调用0次结果一致,
按照RESTful的要求, GET方法也应该是无副作用的,
这些与契约式设计(Design By Contract)的“区分命令与查询”原则是一致的。
但框架在这方面没法做太多限制, 只能在文档中声明并告知业务开发人员。

3. 客户端操作方式

要保证的语义是: 客户端看起来是在操作资源表形本身。


(1) 客户端

首先, 客户端需要一个代表资源的接口, 它的所有函数都是操作该资源本身, 如:

Java代码 ☆

```
1.  /**
2.   * RESTful 远程资源接口
3.   * @param <R> 资源类型
4.   */
5.  public interface Resource<R extends Serializable> {
6.
7.      /**
8.       * 创建资源
9.       * @param resource 资源信息(注: 标识性属性值(如: ID值)无效, 在服务器端接收时, 将被替换为资源URI所指定的值)
10.
11.       * @throws Exception 创建失败或网络连接出错时抛出
12.       */
13.       void create(R resource) throws Exception;
14.
15.       /**
16.        * 更新资源
17.        * @param resource 资源信息(注: 标识性属性值(如: ID值)无效, 在服务器端接收时, 将被替换为资源URI所指定的值)
18.
19.        * @throws Exception 更新失败或网络连接出错时抛出
20.        */
21.        void update(R resource) throws Exception;
22.
23.       /**
24.        * 删除资源
25.        * @throws Exception 删除失败或网络连接出错时抛出
26.        */
27.        void delete() throws Exception;
28.
29.       /**
30.        * 获取资源
31.        * @return 资源
32.        * @throws Exception 获取失败或网络连接出错时抛出
33.        */
34.        R get() throws Exception;
35.    }
```

然后，通过工厂获取此接口，如：

Java代码 


```
1. Resource<User> userResource = Resources.getResource("/users/4271"); // URI带上ID
2. // 或者: Resources.getResource("/users/{0}", id); // 不定长参数, 替换URI中的占位符
3. User user = new User("liangfei", "xxx@xxx.com");
4. userResource.create(user);
5. user = userResource.get();
6. userResource.update(user);
7. userResource.delete();
```

另外，可以考虑增加一个批量资源接口，如：

Java代码 

```
1. /**
2.  * 批量资源接口
3.  * @param <R> 资源类型
4.  */
5. public interface BatchResource<R extends Serializable> extends Resource<R[]> {
6.
7.     // 将从父接口中继承:
8.     // void create(R[] resources) throws Exception; // 批量创建
9.     // void update(R[] resources) throws Exception; // 批量更新
10.    // void delete() throws Exception; // 删除全部资源
11.    // R[] get() throws Exception; // 获取全部资源
12.
13.    /**
14.     * 删除匹配的资源
15.     * @param resource 匹配条件(如果条件复杂, 可以传入资源类型的子类作为条件)
16.     * @throws Exception 删除失败或网络连接出错时抛出
17.     */
18.    void delete(R resource) throws Exception;
19.
20.    /**
21.     * 获取匹配的资源
22.     * @param resource 匹配条件(如果条件复杂, 可以传入资源类型的子类作为条件)
23.     * @return 资源
24.     * @throws Exception 获取失败或网络连接出错时抛出
25.     */
26.    R[] get(R resource) throws Exception;
27.
28. }
```

同样，可以使用工厂获取此接口，如：

Java代码 

```
1. BatchResource<User> userBatchResource = Resources.getBatchResource("/users"); // URI不带ID
2. User[] users = userBatchResource.get();
```

(2) AJAX客户端

尽量保持与RCP客户端一致，如：

Javascript代码 

```
1. var userResource = Resources.getResource("/users/4271");
2. var user = {name:"liangfei", emial: "xxx@xxx.com"};
3. userResource.create(user);
4. user = userResource.get();
5. userResource.update(user);
6. userResource.delete();
```

(3) Web页面



Html代码

```
1. <a href="/users/1">view</a>
2. <form action="/users/1" method="post" accept="text/html">
3.     <input type="text" name="name" />
4.     <input type="text" name="email" />
5.     <input type="submit" value="create"/>
6. </form>
```

4. 资源多重表述

也就是客户端，可以通过修改"Accept"请求头信息，来要求服务器端返回不同类型的数据结果，如：

Accept: text/json 返回JSON数据

Accept: text/xml 返回XML数据

Accept: text/html 返回HTML页面

Accept: text/wml 返回WML页面

这个可以通过读取根据请求头信息来切换序列化器实现，首先需要留有策略接口，如：

Java代码 ☆

```
1. /**
2.  * Action接收映射接口
3.  */
4. public interface ActionMapper {
5.
6.     /**
7.      * 获取序列化器
8.      * @param request 请求信息
9.      * @return 序列化器
10.     */
11.     Serializer getSerializer(HttpServletRequest request);
12.
13.     // 当然，此接口可能还有相关的其它映射函数，如：getActionName(request)等
14.
15. }
```

然后写一个Restful的实现，如：

Java代码 ☆

```
1. public class RestfulActionMapper implements ActionMapper {
2.
3.     private final Map<String, Serializer> serializers = new HashMap<String, Serializer>();
4.
5.     public RestfulActionMapper() {
6.         // 搜索或注册序列化器...
7.         serializers.put("text/json", new JsonSerializer());
8.         serializers.put("text/xml", new XmlSerializer());
9.         serializers.put("text/html", new JspHtmlSerializer());
10.        serializers.put("text/wml", new JspWmlSerializer());
11.        // serializers.put("text/html", new VelocityHtmlSerializer());
12.        // serializers.put("text/wml", new VelocityWmlSerializer());
13.    }
14.
15.    @Override
16.    public Serializer getSerializer(HttpServletRequest request) {
17.        String type = request.getHeader("Accept");
18.        return serializers.get(type);
19.    }
20.
21. }
```

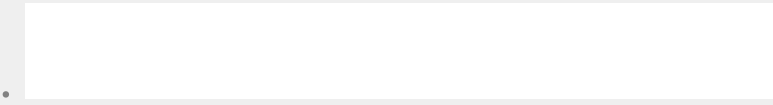
上面提到过结果类型放在"Accept"信息头中有个问题，就是不便输入，对于B/S应用的浏览器来说，不太友好，所以Struts2等通过URL扩展名来识别，如："/user/1.html"，"/user/1.wml"，这样做超接会方便些，但却违反了RESTful语义。

很多细节还没考虑清楚，先写到这了，等正式加入框架，再写个帮助文档。

声明：ITeye 文章版权属于作者，受法律保护。没有作者书面许可不得转载。

推荐链接

- 20-30万急聘多名天才Java/MTA软件工程师
- 女友学不了编程，就叫她来免费学网页、网络、平面、动漫吧！
- .Net程序员不用羡慕Android程序员，微软移动战役打响



返回顶楼



javatar

等级: ☆☆☆☆☆



性别: ♂

文章: 251

积分: 578

来自: 杭州699号



发表时间: 2008-12-08 最后修改: 2008-12-16

经过再三思考，决定使用以下接口形式：

客户端使用：

Java代码 ☆

```
1. // 从工厂中获取资源目录引用（包装代理类，不与服务器交互）
2. Resources<User> userResources = Client.getClient().getResources("/users");
3. // 统计资源列表（发送"HEAD"请求）
4. long userCount = userResources.count();
5. long userCount = userResources.count(new User("xxx"));
6. // 获取资源列表（发送"GET"请求）
7. Resource<User>[] users = userResources.list();
8. Resource<User>[] users = userResources.list(new User("xxx")); // 可以使用User的子类作为过滤条件
9. Resource<User> userResource = users[0];
10. // 创建资源（发送"POST"请求）
11. Resource<User> userResource = userResources.create(new User("liangfei", "xxx@xxx.com"));
12. // 从工厂中获取资源引用（包装代理类，不与服务器交互）
13. Resource<User> userResource = Client.getClient().getResource("/users/{0}", id); // 不定长参数，替换URI中的占位符
14. // 读取资源（发送"GET"请求）
15. User user = userResource.read();
16. // 更新资源（发送"PUT"请求）
17. userResource.update(user);
18. // 删除资源（发送"DELETE"请求）
19. userResource.delete();
```

服务器端处理：

Java代码 ☆

```
1. @Path("/users/{id}") // 双斜杠用于分隔集合资源URI和单一资源URI
2. public class UserResourceAction extends ResourceAction<User> {
3.
4.     // 统计资源列表（接收集合资源的"HEAD"请求）
5.     protected long count(User user) throws Exception {
6.         if (user == null)
7.             return userService.countAllUser();
8.         return userService.countByUser(user);
9.     }
10.
11.     // 获取资源列表（接收集合资源的"GET"请求）
12.     protected User[] list(User user) throws Exception {
13.         if (user == null)
14.             return userService.findAllUser().toArray(new User[0]);
15.         return userService.findByUser(user).toArray(new User[0]);
```



```
16.         }
17.
18.         // 创建资源 (接收集合资源的"POST"请求)
19.         protected void create(User user) throws Exception {
20.             userService.saveUser(user);
21.         }
22.
23.         // 读取资源 (接收单一资源的"GET"请求)
24.         protected User read(User user) throws Exception {
25.             return userService.loadUser(user.getId());
26.         }
27.
28.         // 修改资源 (接收单一资源的"PUT"请求)
29.         protected void update(User user) throws Exception {
30.             userService.updateUser(user);
31.         }
32.
33.         // 删除资源 (接收单一资源的"DELETE"请求)
34.         protected void delete(User user) throws Exception {
35.             userService.removeUser(user.getId());
36.         }
37.
38.     }
```

返回顶楼

 [主页](#)  [资料](#)  [短信](#)  [留言](#)  [关注](#) [回帖地址](#)

 0  0 请登录投票

Beyon_javaeye
等级: 初级会员

发表时间: 2008-12-18

我个人认为用 标注 和 配置文件 灵活性更大些， 比如： get()、 find()、 read()等都可以标注成@Get， 感觉这样隔离了实现！



性别: 

文章: 2

积分: 30

来自: ...

 我现在离线

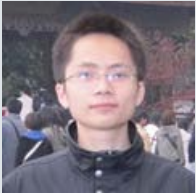
返回顶楼

 [主页](#)  [资料](#)  [短信](#)  [留言](#)  [关注](#) [回帖地址](#)

 0  0 请登录投票

javatar
等级: 

发表时间: 2008-12-18 最后修改: 2008-12-18



Beyon_javaeye 写道

我个人认为用 标注 和 配置文件 灵活性更大些， 比如： get()、 find()、 read()等都可以标注成@Get， 感觉这样隔离了实现！

性别: 

文章: 251

积分: 578

来自: 杭州699号

 我现在离线

是的，用标注是个好想法，[JSR311](#)就是全部用标注实现的，但我觉得REST与RPC的一个重要区别就在于它统一了“动词”，并以“名词”为中心，要的就是不“灵活”(零风格是最灵活的)，虽然会对某些特殊功能实现带来麻烦，但对常见功能非常简化，而且我想实现的是MVC框架，而不是REST版的WebService(这是JSR311的目标)，将REST作为控制器层实现，所以固定函数名，会比较合理些，并且可以保留对"Action"概念的支持，以处理特殊情况。

返回顶楼

 [主页](#)  [资料](#)  [短信](#)  [留言](#)  [关注](#) [回帖地址](#)

 1  0 请登录投票

DanielYWoo

发表时间: 2009-01-14

等级: 初级会员



性别:

文章: 5

积分: 30

来自: 上海

我现在离线

目前HP的Symphony SDK和社区的Restlet和Jersey还算是比较好用,Struts和Axis2所宣称的Restful WS都很糟糕. 另外,我觉得Atom和JSON已经是Restful WS的既定事实的必须支持的两种representation了

引用

如: @Path("/{users/{id}","/users","/users/add","/users/{id}/edit"})

当然,可以用命名约定来减少路径的个数,问题在于add和edit已经超出(甚至违背了)RESTful风格,

如果是自增主键的话,add可以POST到resources上啊,这是Restful WS推荐的方式也是Atom协议要求的. Edit可以put到resources/{id}上

返回顶楼



回帖地址



0



1

请登录投票

[论坛首页](#) → [Java编程和Java企业应用版](#) → [Struts](#)

跳转论坛:

- [四川: 文轩在线诚聘JAVA开发工程师 \(初、中级\)](#)
- [北京: Coplogic诚聘 JAVA高级工程师 \(15万-30万\)](#)
- [北京: 云壤诚聘java中级软件开发工程师 \(J2ee\)](#)
- [北京: 杰嘉迪诚聘高级Java程序员](#)
- [四川: 文轩在线诚聘JAVA开发工程师 \(高级\)](#)
- [上海: 为为网诚聘 JAVA 高级开发工程师](#)

[广告服务](#) | [ITeye黑板报](#) | [联系我们](#) | [友情链接](#)

© 2003-2011 ITeye.com. [[京ICP证110151号](#) [京公网安备110105010620](#)]

百联优力(北京)投资有限公司 版权所有

上海炯耐计算机软件有限公司 提供商务支持