Spring+Hibernate网站开发教程

(V1.0)

作者：刘骥
重庆大学计算机学院
liujiboy@cqu.edu.cn

目录

1.简介

本文以bookshop网站为例，说明基于Spring+Hibernate架构的网站开发方法。

本文所涉及的开发工具包括：

(1)Java SE(http://www.oracle.com/technetwork/java/javase/downloads/index.html)，使用最新的8.0版本。

(2)Eclipse(http://www.eclipse.org/)，集成开发环境使用Eclipse IDE for Java EE Developers最新版本(已经集成了Git和Maven插件)

(3)Git(http://git-scm.com/download/)，版本控制工具，Eclipse已经集成Git插件，但仍然建议安装一款Git客户端。

(4)Maven(http://maven.apache.org)，项目构建工具，Eclipse已经集成Maven插件，建议仍然下载安装Maven。本项目将使用Maven创建的工程模板。

(5)Spring(http://projects.spring.io/spring-framework/)，项目采用的主框架，功能强大，什么都有，但我们只会使用到其中的一部分。

(6)Hibernate(http://hibernate.org)，ORM框架，支持JPA，本项目中用于操作数据库。

(7)JQuery(http://jquery.com)，Javascript框架，能够简化Javascript的开发。

(8)ExtJS(http://www.sencha.com)，Javascript框架，提供一套优美的UI，但速度较慢。相对来说JQuery更轻量级，能用JQuery的地方，尽量用JQuery。

(9)Mysql(http://www.mysql.com)，开源数据库。由于项目使用Hibernate，因此使用什么数据库差异都不大。在开发期使用Mysql的原因在于：Mysql占用的资源比SQL Server和Oracle更少；MySQL用命令行操作可以强化SQL语句的使用，利用大家熟悉SQL语句。

(10)Tomcat(http://tomcat.apache.org)，Web服务器，项目使用7.0以上版本。

2.案例

2.1创建bookshop数据库

首先启动MySQL数据库，然后进入命令行界面(本文的数据库操作全部使用命令行操作，在实际开发中命令行往往比图形化工具更为简单)，输入如下指令：

```
mysql -u root -proot
```

默认情况下mysql的管理员用户为root，密码为root。-u参数说明登录用户为root，-p指令说明root的密码为root。注意-p和root之间没有空格。若设置的密码与本文不同，请相应修改。登录之后界面如下：

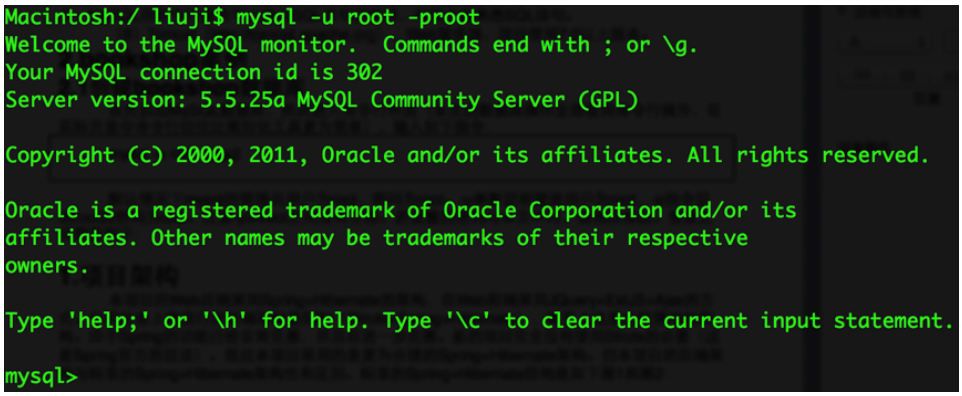


图 1 MySQL命令行界面

接着创建一个名为bookshop的数据库，字符编码默认为utf8，指令如下：

create database bookshop character set=utf8;

数据库中的表不需要创建，一会儿我们会让Hibernate自动创建表。

2.2 导入bookshop项目

启动Eclipse，选择File，选择Import，然后选择Maven，导入Existing Maven Projects

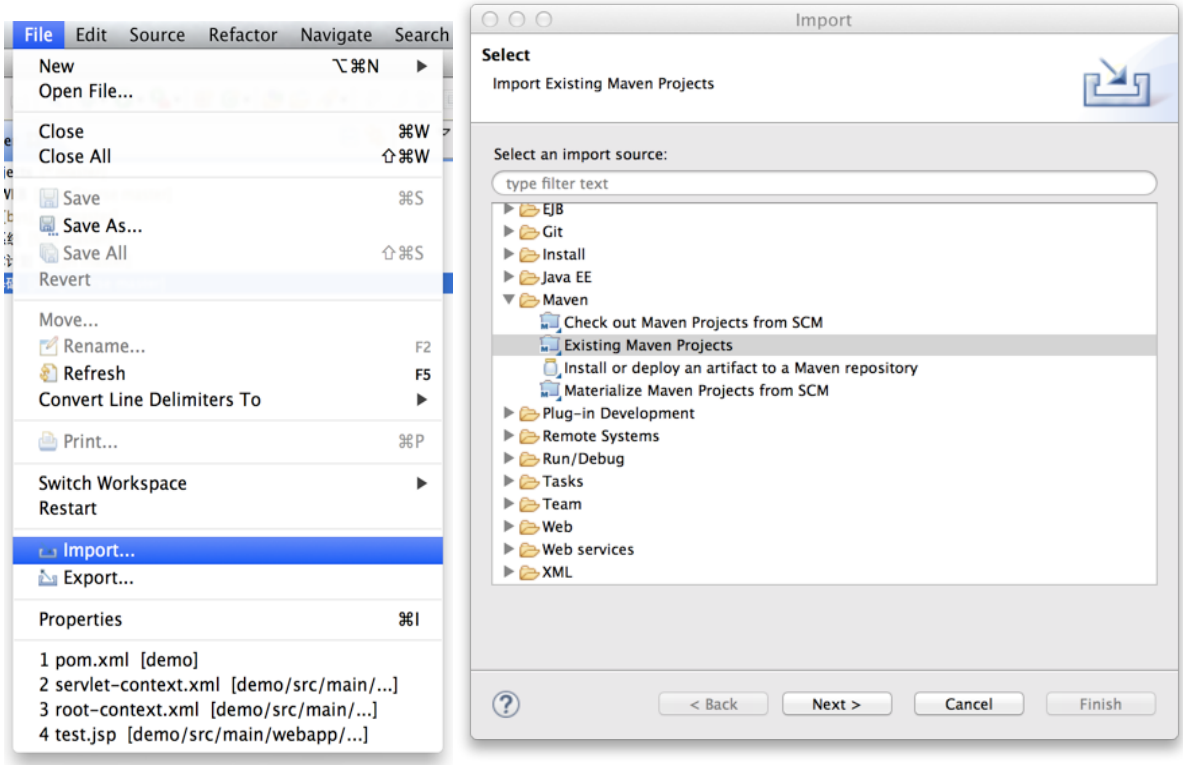


图 2导入bookshop工程

bookshop工程的结构如下：

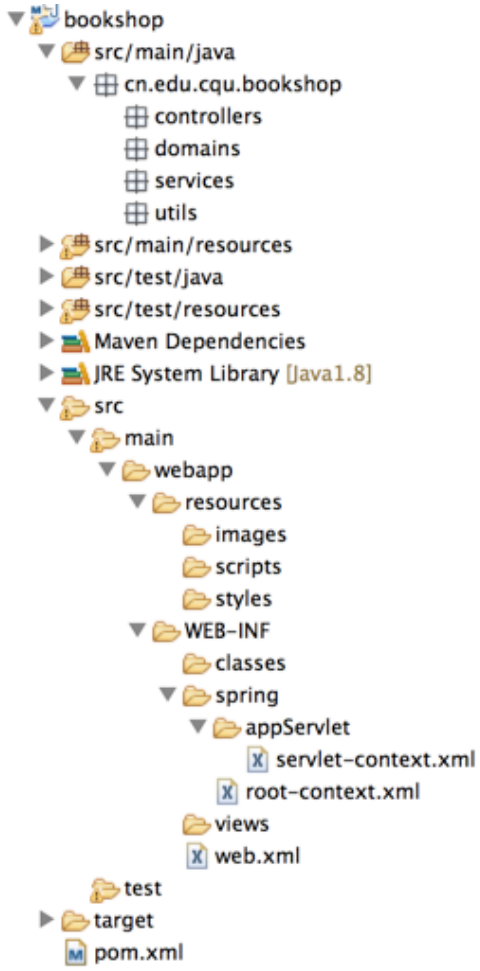


图 3 bookshop工程结构

bookshop工程约定：

(1)程序的源程序全部放在src/main/java目录的cn.edu.cqu.bookshop包之下。其中controllers包放所有的控制器类(处理web请求)，domains放所有的实体类(映射到数据库的表)，services包放所有的业务类(操作数据库处理业务)，utils包放所有的工具类。

(2)所有的视图文件(JSP)放在src/main/webapp/WEB-INF/views目录下。

(3)静态资源文件，如图片、css和Javascript脚本等放在src/main/webapp/resources目录下，并且Web页面上的绝对路径为/resources/**。其中images目录放图片，scripts目录放Javascript脚本，styles目录放css样式表。

(4)servlet-context.xml是spring的配置文件，web.xml是web工程的配置文件，pom.xml是maven的配置文件。

注意：要恢复初始工程可以在命令行下进入工程根目录，然后键入指令：

git checkout v1

或者在Eclipse中切换Tag，具体方法如下图：

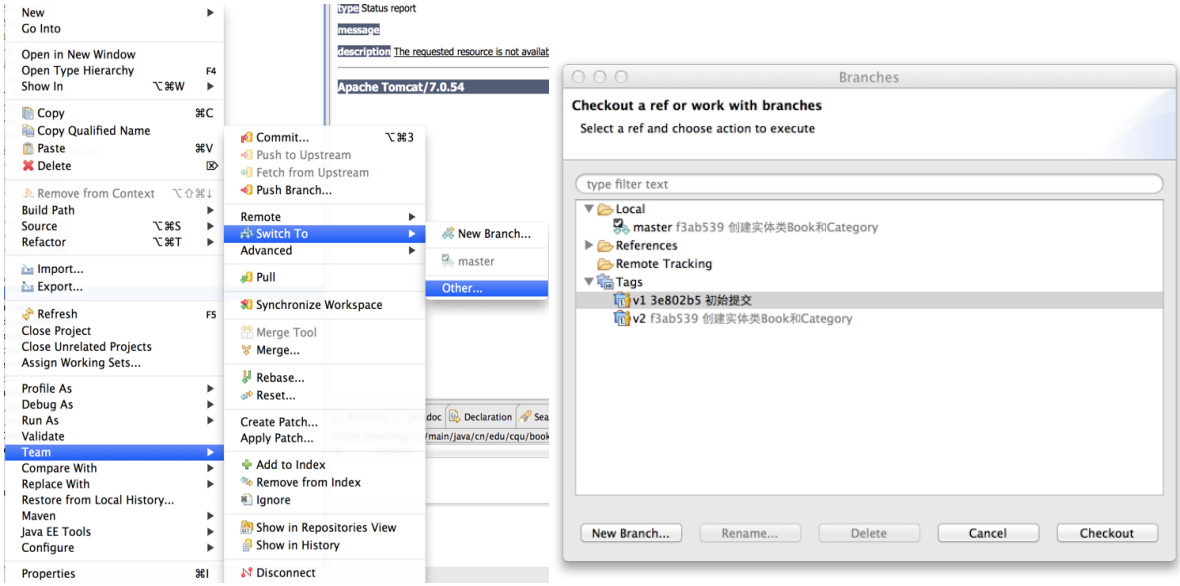


图 4 切换版本

2.3 创建数据表

使用Hibernate可以采用3种创建数据表和实体类的方法[1]：

(1)中间相遇，先用SQL语句创建数据表，再创建实体类，再用Hibernate建立两者之间的关系。

(2)自底向上，先用SQL语句创建数据表，由Hibernate工具将表映射为实体类。

(3)自顶向下，创建实体类，由Hibernate工具将其映射为数据表。

这3种方式中，(1)和(2)常用与已经有数据库的情况，对于全新的数据库推荐采用第(3)种方式。

创建类Category用于描述书籍分类，其代码如下：

```
package cn.edu.cqu.bookshop.domains;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="categories")
public class Category {
	/**
	 * 自动增长的逻辑主键
	 */
	@Id
	@GeneratedValue
	private Long id;
	/**
	 * 类型码(唯一)
	 */
	@Column(length=10,nullable=false,unique=true)
```

[1] Hibernate是一种ORM工具，所谓ORM就是将关系数据库中的表映射到程序中的类。通常可以称这种与数据表对应的类为实体类(entity)，或者领域类(domain)。

```
	private String code;
	/**
	 * 类型名(唯一)
	 */
	@Column(length=100,nullable=false,unique=true)
	private String name;
	public Long getId() {
		return id;

	}
	public void setId(Long id) {
		this.id = id;
	}
	public String getCode() {
		return code;
	}
	public void setCode(String code) {
		this.code = code;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}

}
```

创建类Book用于描述书籍，其代码如下：

```
package cn.edu.cqu.bookshop.domains;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Lob;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
@Entity
@Table(name="books")
public class Book {
	/**
	 * 自动增长的逻辑主键
	 */
```

```java
    @Id
    @GeneratedValue
    private Long id;
    /**
     * 书名(唯一)
     */
    @Column(length=200,nullable=false,unique=true)
    private String name;
    /**
     * 书籍描述
     * 对于长度很大的字符串用lob
     */
    @Lob
    @Basic(fetch = FetchType.LAZY)
    private String info;
    /**
     * 与category的关联
     */
    @ManyToOne(optional=false)
    private Category category;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public String getInfo() {
        return info;
    }
    public void setInfo(String info) {
        this.info = info;
    }
    public Category getCategory() {
        return category;
    }
    public void setCategory(Category category) {
        this.category = category;
    }
```

```
}
```

创建好这两个类之后运行整个工程，选择Run on Server，然后选择Tomcat服务器并运行。若没有任何错误提示信息，那么数据库中就会创建books和categories两个表格。

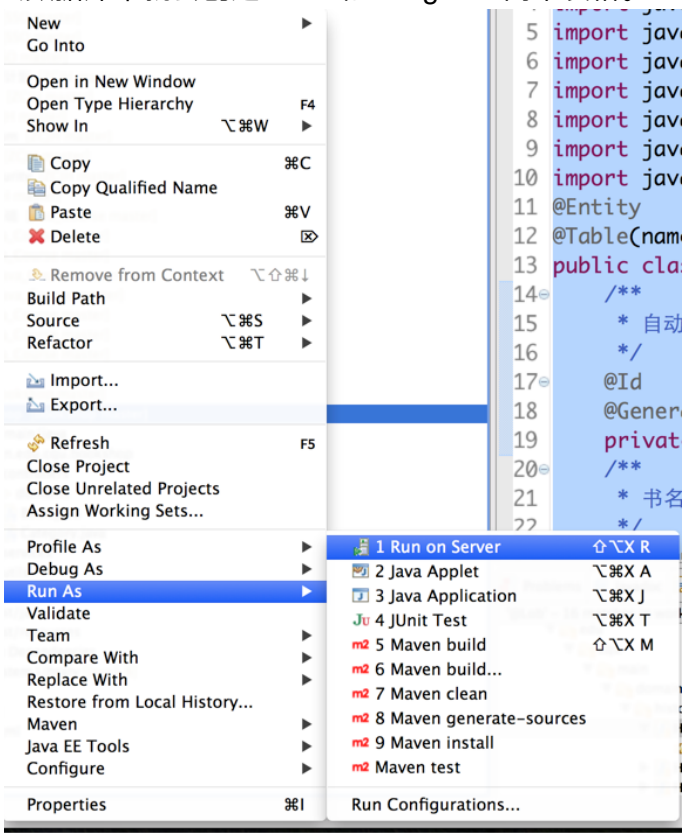


图 5 运行工程

在MySQL中依次录入如下指令查看数据表创建的结果：

```
use bookshop;
show tables;
describe books;
describe categories;
```

结果如下：

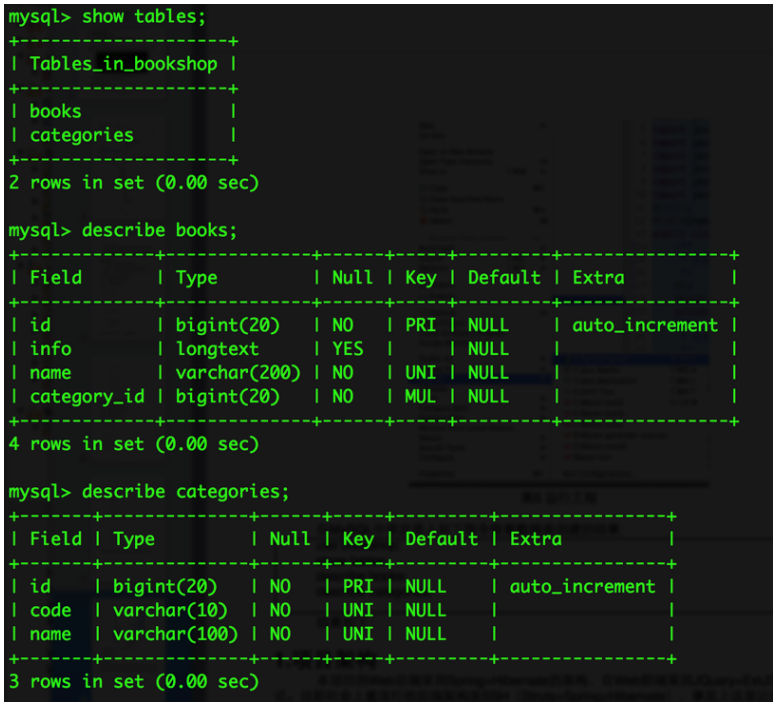


图 6 查看数据库创建情况

从上图可以看出，类Book到Category定义了@ManyToOne关系，在数据库表中就被映射为外键category_id(books表)。@ManyToOne是最常见的类间关系，也是最常见的数据表间关系。

注意：要恢复工程到本小节可以在命令行下进入工程根目录，然后键入指令：

git checkout v2

或者用Eclipse切换到Tag v2

2.4 CRUD业务类

2.4.1 定义CRUD接口

所谓CRUD是指对数据的Create、Read、Update和Delete操作。CRUD是最简单的数据操作。现在我们以categories表为例，实现其CRUD操作。对categories表的CRUD业务方法定义为如下接口：

```
package cn.edu.cqu.bookshop.services;
import java.util.List;
import cn.edu.cqu.bookshop.domains.Category;
public interface CategoryCRUD {
      public Category getById(Long id);
      public void add(Category category);
      public void edit(Long id,Category newCategory);
      public Category getByCode(String code);
      public Category getByName(String name);
      public List<Category> getAll();
      public void deleteById(Long id);
}
```

2.4.2 实现CRUD接口

CategoryCRUD只是一个接口，还需要定义其实现类。定义实现类CategoryCRUDHibernate如下：

```java
package cn.edu.cqu.bookshop.services.hibernate;
import java.util.List;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.orm.hibernate4.HibernateTemplate;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import cn.edu.cqu.bookshop.domains.Category;
import cn.edu.cqu.bookshop.services.CategoryCRUD;

@Service("categoryCRUDHibernate")
public class CategoryCRUDHibernate implements CategoryCRUD {
	private HibernateTemplate ht;

	@Autowired
	public void setSessionFactory(SessionFactory sessionFactory) {
		ht = new HibernateTemplate(sessionFactory);
	}

	@Override
	public Category getById(Long id) {
		return ht.get(Category.class, id);
	}

	@Override
	@Transactional
	public void add(Category category) {
		ht.save(category);
	}

	@Override
	@Transactional
	public void edit(Long id, Category newCategory) {
		Category c = this.getById(id);
		c.setCode(newCategory.getCode());
		c.setName(newCategory.getName());
		ht.update(c);
	}

	@SuppressWarnings("unchecked")
	@Override
	public Category getByCode(String code) {
		List<Category> categories = (List<Category>) ht.find(
				"from Category where code=?", code);
		if (categories.size() > 0)
			return categories.get(0);
```

```
            else
                  return null;
      }

      @SuppressWarnings("unchecked")
      @Override
      public Category getByName(String name) {
            List<Category> categories = (List<Category>) ht.find(
                        "from Category where name=?", name);
            if (categories.size() > 0)
                  return categories.get(0);
            else
                  return null;
      }

      @SuppressWarnings("unchecked")
      @Override
      public List<Category> getAll() {
            List<Category> categories = (List<Category>) ht.find("from
Category");
            return categories;
      }

      @Override
      @Transactional
      public void deleteById(Long id) {
            Category c = this.getById(id);
            ht.delete(c);
      }

}
```

在上述代码中有许多@开头的代码，这种代码称为Annotation。当Spring框架扫描到这些Annotation时，它就会执行特定的操作，这些操作包括：

(1)@Service(“categoryCRUDHibernate”)，用类CategoryCRUDHibernate创建一个名称为categoryCRUDHibernate的对象。

(2)@Autowired，自动进行以来关系注入。下面的代码

```
  @Autowired
      public void setSessionFactory(SessionFactory sessionFactory) {
            ht = new HibernateTemplate(sessionFactory);
      }
```

意味着Spring框架自动调用setSessionFactory这个方法，并将sessionFactory传递到函数中。sessionFactory定义在src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml文件中。换言之，当看到@Autowired时，Spring会自动将ht(类型为HibernateTemplate)创建出来。HibernateTemplate封装了Hibernate的常用方法。

(3)@Transactional，意味着这个函数是一个事务函数，只要函数执行过程中出现错误，任何写操作都会回滚。注意只要函数存在写操作就应该声明为@Transactional。

2.4.3 创建测试用例

CRUD程序创建好之后，必须经过测试，确保其正确，才能编写Web代码。这样做的目的是为了尽早发现错误，避免错误累计。测试的方法很多，常见的方法是采用Junit进行测试。测试代码编写在src/test/java目录下，测试代码如下：

```
package cn.edu.cqu.bookshop.services.hibernate;
import static org.junit.Assert.*;
import java.util.List;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import cn.edu.cqu.bookshop.domains.Category;
import cn.edu.cqu.bookshop.services.CategoryCRUD;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration({"file:src/main/webapp/WEB-INF/spring/appServlet/
servlet-context.xml"})
public class CategoryCRUDHibernateTest {
      @Autowired
      @Qualifier("categoryCRUDHibernate")
      private CategoryCRUD categoryCRUD;
      @Before
    public void setUp() throws Exception {
            //删除所有的数据
            deleteAll();
    }
      private void deleteAll() {
            List<Category> categories=categoryCRUD.getAll();
            for(Category c:categories)
            {
                  categoryCRUD.deleteById(c.getId());
            }
      }
      @After
    public void tearDown() throws Exception {
            deleteAll();
    }
      @Test
      public void testGetById() {
```

```
        Category c=new Category();
        c.setCode("1");
        c.setName("1");
        categoryCRUD.add(c);
        c=categoryCRUD.getById(c.getId());
        assertEquals("1",c.getName());
        assertEquals("1",c.getCode());
    }

    @Test
    public void testAdd() {
        Category c=new Category();
        c.setCode("1");
        c.setName("1");
        categoryCRUD.add(c);
        c=categoryCRUD.getById(c.getId());
        assertEquals("1",c.getName());
        assertEquals("1",c.getCode());
        c=new Category();
        c.setCode("1");
        c.setName("1");
        try{
            categoryCRUD.add(c);
            fail("不能执行到此处");
        }catch(Exception e)
        {

        }
    }

    @Test
    public void testEdit() {
        Category c=new Category();
        c.setCode("1");
        c.setName("1");
        categoryCRUD.add(c);
        c=categoryCRUD.getById(c.getId());
        c.setName("2");
        categoryCRUD.edit(c.getId(), c);
        c=categoryCRUD.getById(c.getId());
        assertEquals("2",c.getName());

    }

    @Test
    public void testGetByCode() {
```

```
            Category c=new Category();
            c.setCode("1");
            c.setName("1");
            categoryCRUD.add(c);
            Category c1=categoryCRUD.getByCode("1");
            assertEquals("1",c1.getName());
      }

      @Test
      public void testGetByName() {
            Category c=new Category();
            c.setCode("1");
            c.setName("1");
            categoryCRUD.add(c);
            Category c1=categoryCRUD.getByName("1");
            assertEquals("1",c1.getCode());
      }

      @Test
      public void testGetAll() {
            Category c1=new Category();
            c1.setCode("1");
            c1.setName("1");
            categoryCRUD.add(c1);
            Category c2=new Category();
            c2.setCode("2");
            c2.setName("2");
            categoryCRUD.add(c2);
            List<Category> categories=categoryCRUD.getAll();
            assertEquals("1",categories.get(0).getName());
            assertEquals("2",categories.get(1).getName());
      }

      @Test
      public void testDeleteById() {
            Category c1=new Category();
            c1.setCode("1");
            c1.setName("1");
            categoryCRUD.add(c1);
            categoryCRUD.deleteById(c1.getId());
            Category c2=categoryCRUD.getById(c1.getId());
            assertEquals(null,c2);
      }
}
```

在上述代码中

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration({"file:src/main/webapp/WEB-INF/spring/appServlet/
servlet-context.xml"})
```

指定了Spring配置文件的位置。

```
    @Autowired
    @Qualifier("categoryCRUDHibernate")
    private CategoryCRUD categoryCRUD;
```

表示categoryCRUD自动设置为名称为categoryCRUDHibernate的对象(请参考上一节谈到的@Service定义)。

每一个@Test Annotation定义了一个测试。出于演示目的，上述测试代码并不能完整覆盖所有情况。最后执行测试程序，打绿色勾的表示测试通过，而打红色叉的表示测试未通过，如下图：

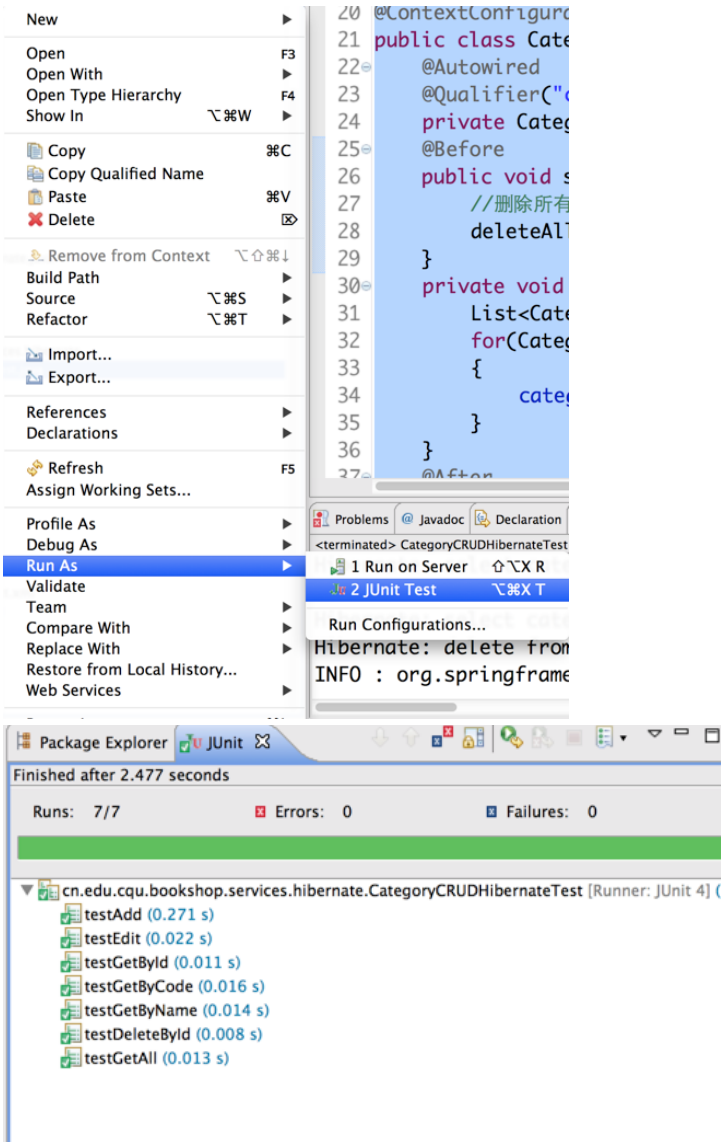


图 7 运行测试

注意：要恢复工程到本小节可以在命令行下进入工程根目录，然后键入指令：
git checkout v3
或者用Eclipse切换到Tag v3

2.4.4 CRUD接口的JDBC实现及其测试

现在我们有一个疑问：定义CategoryCRUD接口，然后再用CategoryCRUDHibernate类实现这个接口，不是多此一举吗？在多数情况下，这的确是多此一举。但如果设想这样一种场景：最初CRUD程序是用Hibernate实现的，但由于某种原因需要将Hibernate替换为JDBC，那么应该怎么做呢？很显然，在有CategoryCRUD接口的情况下，可以实现另一个类CategoryCRUDJDBC，代码如下：

```
package cn.edu.cqu.bookshop.services.jdbc;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.List;
import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.PreparedStatementCreator;
import org.springframework.jdbc.core.ResultSetExtractor;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.support.GeneratedKeyHolder;
import org.springframework.jdbc.support.KeyHolder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import cn.edu.cqu.bookshop.domains.Category;
import cn.edu.cqu.bookshop.services.CategoryCRUD;

@Service("categoryCRUDJDBC")
public class CategoryCRUDJDBC implements CategoryCRUD {
	private JdbcTemplate jt;

	@Autowired
	public void setDataSource(DataSource dataSource) {
		jt = new JdbcTemplate(dataSource);
	}

	@Override
	public Category getById(Long id) {
		return jt.query("select * from categories where id=?",
				new ResultSetExtractor<Category>() {

					@Override
					public Category extractData(ResultSet rs)
```

```
                                        throws SQLException,
DataAccessException {
                                  if (rs.next()) {
                                        Category c = new Category();
                                        c.setCode(rs.getString("code"));
                                        c.setName(rs.getString("name"));
                                        c.setId(rs.getLong("id"));
                                        return c;
                                  } else {
                                        return null;
                                  }
                            }

                      }, id);

      }

      @Override
      @Transactional
      public void add(Category category) {
            KeyHolder keyHolder = new GeneratedKeyHolder();
            final String name = category.getName();
            final String code = category.getCode();
            jt.update(new PreparedStatementCreator() {

                  @Override
                  public PreparedStatement
createPreparedStatement(Connection con)
                              throws SQLException {
                        PreparedStatement ps = con.prepareStatement(
                                    "insert into categories(name,code)
values(?,?)",
                                    Statement.RETURN_GENERATED_KEYS);
                        ps.setString(1, name);
                        ps.setString(2, code);
                        return ps;
                  }
            }, keyHolder);
            category.setId(keyHolder.getKey().longValue());
      }

      @Override
      @Transactional
      public void edit(Long id, Category newCategory) {
            jt.update("update categories set name=?,code=? where
id=?",newCategory.getName(),newCategory.getCode(),id);
```

```
	}

	@Override
	public Category getByCode(String code) {
		return jt.query("select * from categories where code=?",
				new ResultSetExtractor<Category>() {

					@Override
					public Category extractData(ResultSet rs)
							throws SQLException,
DataAccessException {
						if (rs.next()) {
							Category c = new Category();
							c.setCode(rs.getString("code"));
							c.setName(rs.getString("name"));
							c.setId(rs.getLong("id"));
							return c;
						} else {
							return null;
						}
					}

				}, code);
	}

	@Override
	public Category getByName(String name) {
		return jt.query("select * from categories where name=?",
				new ResultSetExtractor<Category>() {

					@Override
					public Category extractData(ResultSet rs)
							throws SQLException,
DataAccessException {
						if (rs.next()) {
							Category c = new Category();
							c.setCode(rs.getString("code"));
							c.setName(rs.getString("name"));
							c.setId(rs.getLong("id"));
							return c;
						} else {
							return null;
						}
					}

				}, name);
```

```
	}

	@Override
	public List<Category> getAll() {
		return jt.query("select * from categories ", new
RowMapper<Category>() {

			@Override
			public Category mapRow(ResultSet rs, int rowNum)
					throws SQLException {
				Category c = new Category();
				c.setCode(rs.getString("code"));
				c.setName(rs.getString("name"));
				c.setId(rs.getLong("id"));
				return c;
			}

		});
	}

	@Override
	@Transactional
	public void deleteById(Long id) {
		jt.update("delete from categories where id=?", id);
	}
}
```

对应的测试程序如下：

```
package cn.edu.cqu.bookshop.services.jdbc;

import static org.junit.Assert.*;

import java.util.List;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import cn.edu.cqu.bookshop.domains.Category;
import cn.edu.cqu.bookshop.services.CategoryCRUD;
@RunWith(SpringJUnit4ClassRunner.class)
```

```
@ContextConfiguration({"file:src/main/webapp/WEB-INF/spring/appServlet/
servlet-context.xml"})
public class CategoryCRUDJDBCTest {
      @Autowired
      @Qualifier("categoryCRUDJDBC")
      private CategoryCRUD categoryCRUD;
      @Before
    public void setUp() throws Exception {
            //删除所有的数据
            deleteAll();

    }
      private void deleteAll() {
            List<Category> categories=categoryCRUD.getAll();
            for(Category c:categories)
            {
                  categoryCRUD.deleteById(c.getId());
            }
      }
      @After
    public void tearDown() throws Exception {
            deleteAll();
    }
      @Test
      public void testGetById() {
            Category c=new Category();
            c.setCode("1");
            c.setName("1");
            categoryCRUD.add(c);
            c=categoryCRUD.getById(c.getId());
            assertEquals("1",c.getName());
            assertEquals("1",c.getCode());
      }

      @Test
      public void testAdd() {
            Category c=new Category();
            c.setCode("1");
            c.setName("1");
            categoryCRUD.add(c);
            c=categoryCRUD.getById(c.getId());
            assertEquals("1",c.getName());
            assertEquals("1",c.getCode());
            c=new Category();
            c.setCode("1");
            c.setName("1");
```

```
            try{
                  categoryCRUD.add(c);
                  fail("不能执行到此处");
            }catch(Exception e)
            {

            }
      }

      @Test
      public void testEdit() {
            Category c=new Category();
            c.setCode("1");
            c.setName("1");
            categoryCRUD.add(c);
            c=categoryCRUD.getById(c.getId());
            c.setName("2");
            categoryCRUD.edit(c.getId(), c);
            c=categoryCRUD.getById(c.getId());
            assertEquals("2",c.getName());

      }

      @Test
      public void testGetByCode() {
            Category c=new Category();
            c.setCode("1");
            c.setName("1");
            categoryCRUD.add(c);
            Category c1=categoryCRUD.getByCode("1");
            assertEquals("1",c1.getName());
      }

      @Test
      public void testGetByName() {
            Category c=new Category();
            c.setCode("1");
            c.setName("1");
            categoryCRUD.add(c);
            Category c1=categoryCRUD.getByName("1");
            assertEquals("1",c1.getCode());
      }

      @Test
      public void testGetAll() {
            Category c1=new Category();
```

```
            c1.setCode("1");
            c1.setName("1");
            categoryCRUD.add(c1);
            Category c2=new Category();
            c2.setCode("2");
            c2.setName("2");
            categoryCRUD.add(c2);
            List<Category> categories=categoryCRUD.getAll();
            assertEquals("1",categories.get(0).getName());
            assertEquals("2",categories.get(1).getName());
      }

      @Test
      public void testDeleteById() {
            Category c1=new Category();
            c1.setCode("1");
            c1.setName("1");
            categoryCRUD.add(c1);
            categoryCRUD.deleteById(c1.getId());
            Category c2=categoryCRUD.getById(c1.getId());
            assertEquals(null,c2);
      }
}
```

于是整个程序呈现出如下的结构：

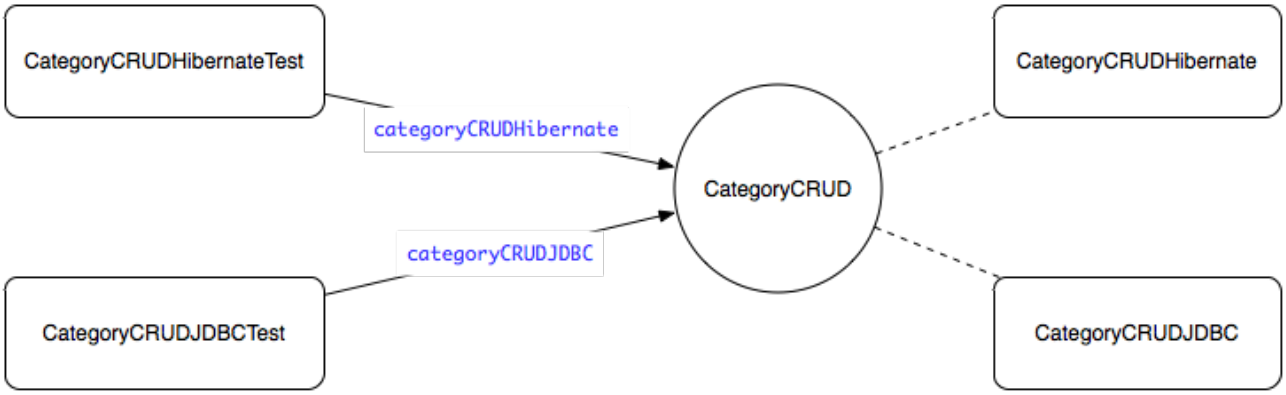


图 8 CRUD程序结构

CategoryCRUDHibernateTest和CategoryCRUDJDBCTest只依赖于CategoryCRUD接口，而不依赖于实现接口的具体类。通过Spring的依赖注入，CategoryCRUDHibernateTest和CategoryCRUDJDBCTest通过@Autowired和@Qualifier注入正确的依赖关系。通过观察我们很容易发现CategoryCRUDHibernateTest和CategoryCRUDJDBCTest的代码几乎一模一样，除了@Qualifier的值。@Qualifier的值为categoryCURDHibernate则注入CategoryCURDHibernate类，反之注入CategoryCRUDJDBC。

故此“定义CategoryCRUD接口，再定义其实现类”的原因在于：

(1)便于编写测试程序，测试程序仅依赖于CategoryCRUD接口。

(2)可通过Spring的依赖注入功能，切换不同的CategoryCRUD接口实现。例如可以提供CategoryCRUDHibernate和CategoryCRUDJDBC两种实现，然后根据实际情况进行选择。

当然上述做法也有显而易见的副作用，那就是程序结构变得更复杂了。如果程序不存在上述两种需求(没有专门的测试团队编写测试程序，没有从Hibernate和JDBC间切换的需求)，那么上述结构无疑是多余的。

注意：要恢复工程到本小节可以在命令行下进入工程根目录，然后键入指令：
git checkout v4
或者用Eclipse切换到Tag v4

2.5 CRUD Web程序

2.5.1 MVC模式

目前Web程序都会采用MVC模式，其结构如下图所示：

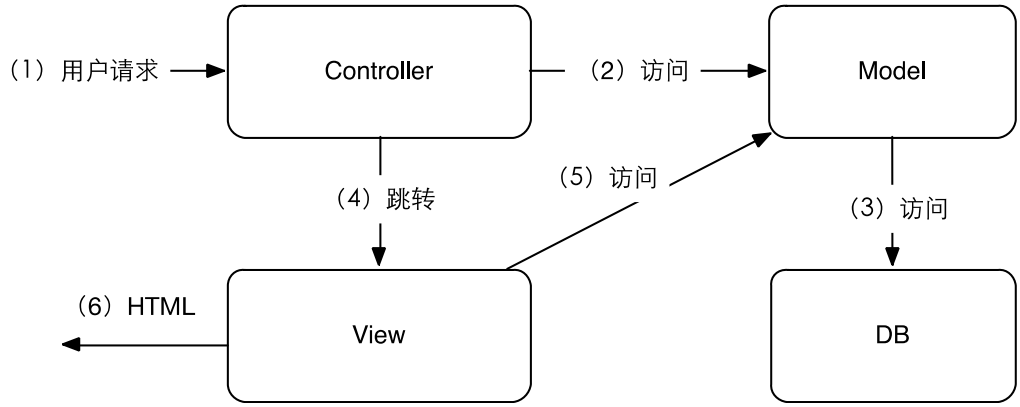


图 9 MVC模式

对很多人而言，上图略显抽象，下面举一个更为具体的例子。假设用户通过浏览器请求URL地址/category/listAll.do，该地址显示一个HTML页面列出所有的category，其过程如下图所示：

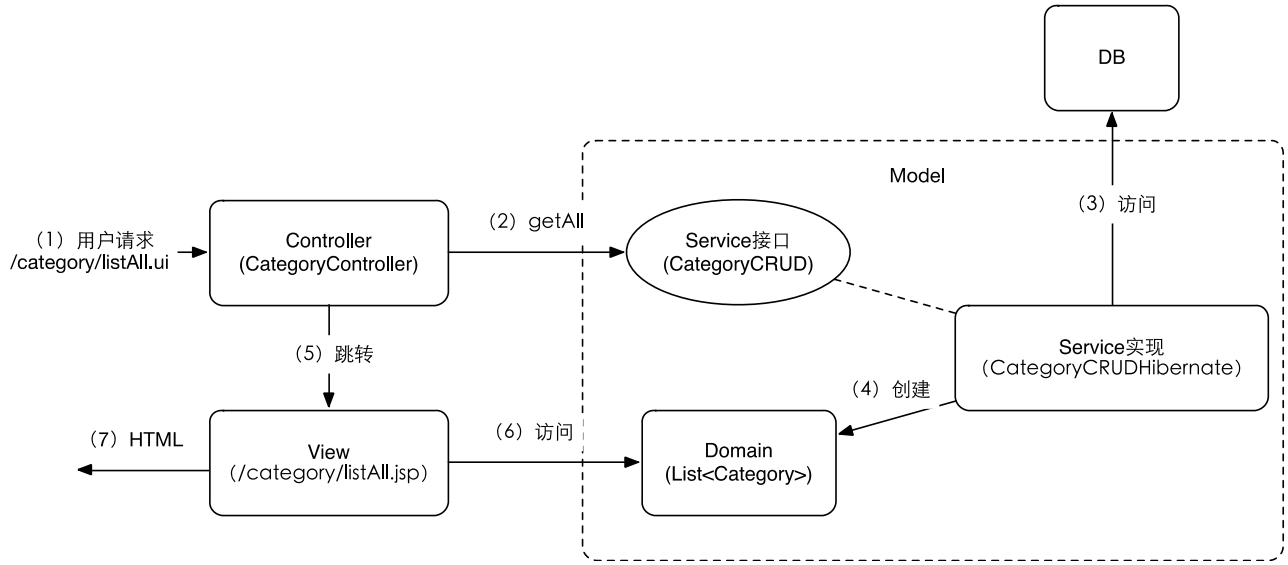


图 10 Spring中的MVC

其中CategoryCRUD、CategoryCRUDHibernate，以及Category等都称为Model。所谓控制器Controller，其主要功能是拦截用户请求，调用业务方法，根据业务方法的返回结果，跳转到对应的视图View。最后视图View根据Model的数据构建HTML页面。

在MVC模式之下，各部分的主要职责为：

(1)Model，实现业务方法，既不依赖Controller也无依赖View(从图 10中箭头的方向就可以得出这个显然的结论)，换言之它完全可以独立编写。

(2)View，实现表示层逻辑，依赖于Model(表示数据的那些Model，例如Category)，不依赖于Controller。需要注意HTML页面仅仅是View的一种形式，View也可以是其他形式。更不能将JSP等同为View，尽管JSP是View的一种常见形式。

(3)Controller，处理Web请求，整合Model和View。Controller主要处理与Web相关的事物，包括权限管理、数据有效性验证、异常处理、调用Model、跳转到View和向View传递数据等。整个Web程序中，Controller的功能最为繁琐，但很难体现技术含量。

MVC模式的优势在于，Controller、Model和View之间耦合度较低，因而可以实现Controller、Model和View的并行开发。事实上在没有实现任何Controller和View之前，我们已经完成了categories数据表CRUD操作的全部Model，并且进行了测试。页面美工也可以在预先设计好视图，最后用Controller就可以将Model和View联系起来。

2.5.2 实现DemoController

下面我们来编写一个简单的Controller。新建一个类DemoController，代码如下：

```
package cn.edu.cqu.bookshop.controllers;

import java.util.HashMap;
import java.util.Map;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.servlet.ModelAndView;

import cn.edu.cqu.bookshop.domains.Category;

@Controller
public class DemoController {
	@RequestMapping(value="/demo/hello.do")
	public ModelAndView hello(String name)
	{
		ModelAndView mv=new ModelAndView();
		if(name==null||name.equals(""))
		{
			name="world";
		}
		mv.addObject("name",name);
		mv.setViewName("/demo/hello");
		return mv;
	}
	@RequestMapping(value="/demo/map.json")
	@ResponseBody
	public Map<String,String> mapJson()
	{
		Map<String,String> map=new HashMap<String,String>();
		map.put("message", "hello world!");
		map.put("type","just for demo" );
```

```
            return map;
      }
      @RequestMapping(value="/demo/category.json")
      @ResponseBody
      public Category category()
      {
            Category c=new Category();
            c.setCode("123");
            c.setName("abc");
            return c;
      }
}
```

在上述代码中：

(1)@Controller表示DemoController是一个Controller类，Spring会对其进行管理。

(2)@RequestMapping将方法hello(String name)映射到URL地址/demo/hello.do。参数name对应浏览器传递过来的名称为name的数据，例如/demo/hello.do?name=liuji，则参数name的值为liuji。

(3)@RequestMapping将方法mapJson()映射到URL地址/demo/map.json。

(4)@RequestMapping将方法category()映射到URL地址/demo/category.json。

(5)@ResponseBody表示该方法以JSON格式返回数据。

(6)hello(String name)返回ModelView类型，该类型用addObject方法添加数据(可在视图上依据名称获取)，用setView方法设置跳转视图。

然后在src/main/webapp/WEB-INF/views/demo目录下(需要新建demo目录)创建文件hello.jsp，其代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!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>demo</title>
</head>
<body>
hello ${name}!
</body>
</html>
```

其中${name}用于访问ModelView中的名称为name的数据。

下面运行工程：

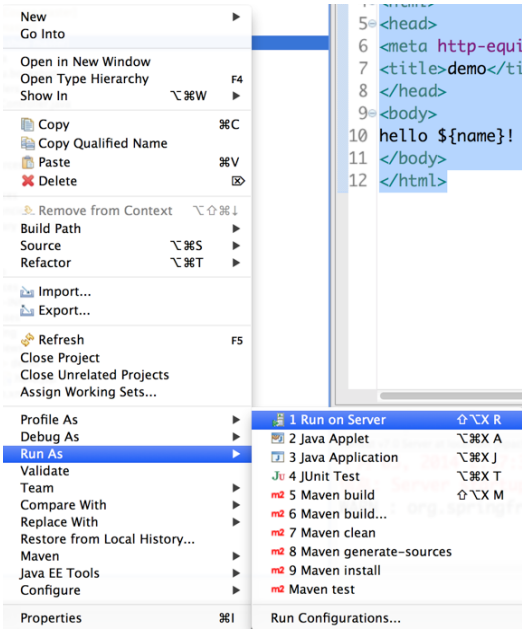


图 11 运行Web工程

之后在浏览器中依此访问如下链接：

http://localhost:8080/bookshop/demo/hello.do
http://localhost:8080/bookshop/demo/hello.do?name=liuji
http://localhost:8080/bookshop/demo/map.json
http://localhost:8080/bookshop/demo/category.json

显示结果如下：

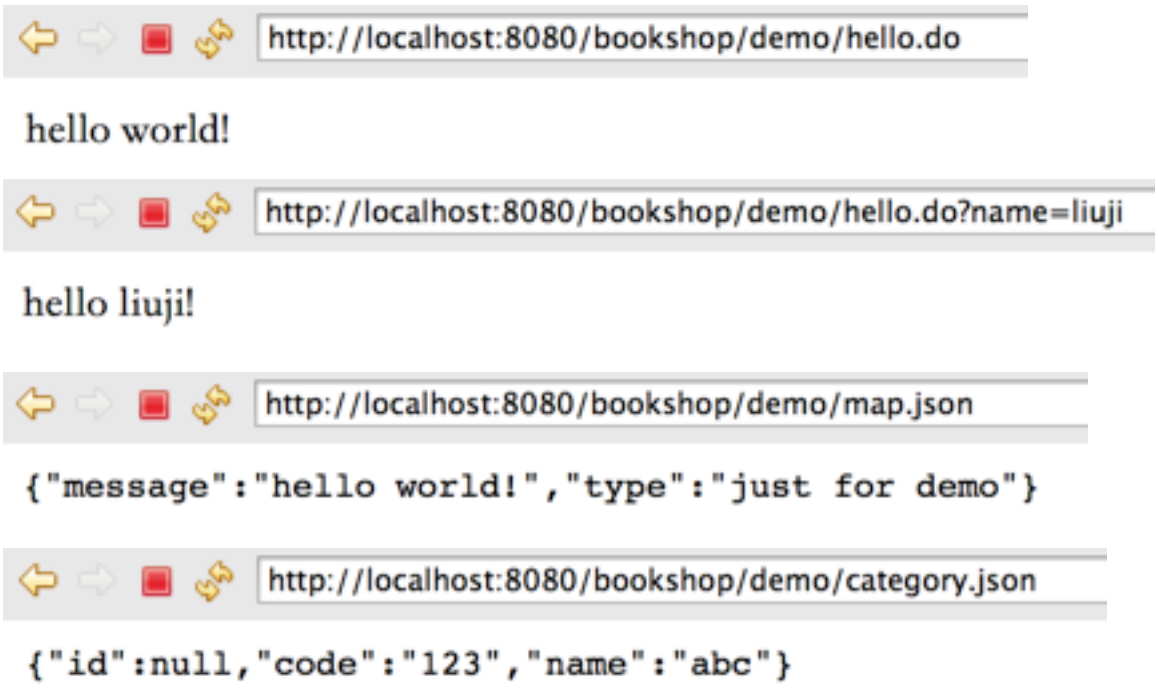


图 12 DemoController的显示结果

通过DemoController可以得出以下几点结论：

(1)在MVC模式下不能直接访问视图，视图只能通过Controller跳转，或者说任何视图都位于Controller之后，即使这个Controller什么也不做。

(2)ModelView用于跳转视图(setViewName)，并向视图传递数据(addObject)。
(3)在JSP视图上，通过${XX}可以访问Controller传递过来的数据。
(4)方法声明@ResponseBody，可以返回JSON格式的数据。
注意：要恢复工程到本小节可以在命令行下进入工程根目录，然后键入指令：
git checkout v5
或者用Eclipse切换到Tag v5

2.5.3 实现CategoryController

现在我们来实现添加Category的Web程序。添加Category包含至少两个Controller方法，一个方法用于显示添加Category的页面(/category/add.ui)，另一个方法用于执行添加Category的操作并根据执行结果进行跳转(/category/add.do)。为了显示添加Category的执行结果，还增加一个方法用于显示所有的category(/category/listAll.ui)，我们增加一个CategoryController，其代码如下：

```
package cn.edu.cqu.bookshop.controllers;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import cn.edu.cqu.bookshop.domains.Category;
import cn.edu.cqu.bookshop.services.CategoryCRUD;

@Controller
public class CategoryController {
	@Autowired
	@Qualifier("categoryCRUDHibernate")
	private CategoryCRUD categoryCRUD;
	@RequestMapping(value="/category/add.ui")
	public ModelAndView addCategoryUI()
	{
		ModelAndView mv=new ModelAndView();
		mv.setViewName("/category/addCategory");
		return mv;
	}
	@RequestMapping(value="/category/add.do")
	public ModelAndView doAddCategory(String code,String name)
	{
		ModelAndView mv=new ModelAndView();
		if(isValid(code,name,mv))
		{
			Category c=new Category();
			c.setCode(code);
			c.setName(name);
```

```
                categoryCRUD.add(c);
                mv.setViewName("redirect:/category/listAll.ui");
            }else
            {
                mv.addObject("code", code);
                mv.addObject("name",name);
                mv.setViewName("/category/addCategory");
            }
            return mv;
        }

        private boolean isValid(String code, String name, ModelAndView mv)
{
            boolean valid=true;
            //检查code不能为空，且长度不能大于10字节
            if(code==null||code.equals(""))
            {
                mv.addObject("codeError", "code不能为空");
                valid=false;
            }else
            {
                if(code.getBytes().length>10)
                {
                    mv.addObject("codeError", "code的长度不能大于10字节");
                    valid=false;
                }
            }
            //检查name不能为空，且长度不能大于100字节
            if(name==null||name.equals(""))
            {
                mv.addObject("nameError", "name不能为空");
                valid=false;
            }else
            {
                if(name.getBytes().length>100)
                {
                    mv.addObject("nameError", "name的长度不能大于100字节
");
                    valid=false;
                }
            }
            if(valid)
            {
                //检查不能存在相同的code
                if(categoryCRUD.getByCode(code)!=null)
```

```
                {
                        mv.addObject("codeError", "code重复");
                        valid=false;
                }
                //检查不能存在相同的nam
                if(categoryCRUD.getByName(name)!=null)
                {
                        mv.addObject("nameError", "name重复");
                        valid=false;
                }
        }
        return valid;
    }
    @RequestMapping(value="/category/listAll.ui")
    public ModelAndView listAll()
    {
        ModelAndView mv=new ModelAndView();
        List<Category> categories=categoryCRUD.getAll();
        mv.addObject("categories",categories);
        mv.setViewName("/category/listAll");
        return mv;
    }
}
```

在上述代码中doAddCategory对数据进行了有效性测试(isValid)，如果测试不通过，就返回错误，并且跳转到输入页面重新输入数据。

与业务类相同，我们也可以对CategoryController进行单元测试，测试代码如下：

```
package cn.edu.cqu.bookshop.controllers;

import static org.junit.Assert.*;

import java.util.List;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.web.servlet.ModelAndView;

import cn.edu.cqu.bookshop.domains.Category;
import cn.edu.cqu.bookshop.services.CategoryCRUD;
```

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration({"file:src/main/webapp/WEB-INF/spring/appServlet/
servlet-context.xml"})
public class CategoryControllerTest {
	@Autowired
	@Qualifier("categoryCRUDHibernate")
	private CategoryCRUD categoryCRUD;
	@Autowired
	private CategoryController categoryController;
	@Before
	public void setUp() throws Exception {
		//删除所有的数据
		deleteAll();
	}
	private void deleteAll() {
		List<Category> categories=categoryCRUD.getAll();
		for(Category c:categories)
		{
			categoryCRUD.deleteById(c.getId());
		}
	}
	@After
	public void tearDown() throws Exception {
		deleteAll();
	}

	@Test
	public void testAddCategoryUI() {
		ModelAndView mv=categoryController.addCategoryUI();
		assertEquals("/category/addCategory",mv.getViewName());
	}

	@Test
	public void testDoAddCategory() {
		Category c=new Category();
		c.setCode("1");
		c.setName("2");
		categoryCRUD.add(c);
		//测试code为空
		ModelAndView mv=categoryController.doAddCategory("", "2");
		assertEquals("/category/addCategory",mv.getViewName());
		assertEquals("code不能为空",mv.getModelMap().get("codeError"));
		//测试code超长
		mv=categoryController.doAddCategory("111111111111", "2");
		assertEquals("/category/addCategory",mv.getViewName());
```

```
            assertEquals("code的长度不能大于10字节
",mv.getModelMap().get("codeError"));
            //测试存在相同的code
            mv=categoryController.doAddCategory("1", "3");
            assertEquals("/category/addCategory",mv.getViewName());
            assertEquals("code重复",mv.getModelMap().get("codeError"));
            //测试name为空
            mv=categoryController.doAddCategory("2", "");
            assertEquals("/category/addCategory",mv.getViewName());
            assertEquals("name不能为空",mv.getModelMap().get("nameError"));
            //测试name超长
            mv=categoryController.doAddCategory("1",
"11111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111");
            assertEquals("/category/addCategory",mv.getViewName());
            assertEquals("name的长度不能大于100字节
",mv.getModelMap().get("nameError"));
            //测试存在相同的name
            mv=categoryController.doAddCategory("2", "2");
            assertEquals("/category/addCategory",mv.getViewName());
            assertEquals("name重复",mv.getModelMap().get("nameError"));
            //测试正常添加
            mv=categoryController.doAddCategory("4", "5");
            assertEquals("redirect:/category/
listAll.ui",mv.getViewName());
            //其他测试...
      }

      @Test
      public void testListAll() {
            ModelAndView mv=categoryController.listAll();
            assertEquals("/category/listAll",mv.getViewName());
      }

}
```

对于有一定规模的程序而言，测试是必须的。测试不仅可以帮助程序员及早发现程序漏洞，同时也可以保证程序模块之间具有低耦合度(对多数程序而言，低耦合度总是好的)。

注意：要恢复工程到本小节可以在命令行下进入工程根目录，然后键入指令：
git checkout v6
或者用Eclipse切换到Tag v6

2.5.4实现View

CategoryController中跳转的View只有一个名字，现在我们来实现这些View。/category/addCategory对应到src/main/webapp/WEB-INF/category/addCategory.jsp，其代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
```

```
    pageEncoding="UTF-8"%>
<!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>添加Category</title>
</head>
<body>
<form action="add.do" method="POST">
<p>name:<input type="text" name="name" value="${name }">${nameError}</p>
<p>code:<input type="text" name="code" value="${code }">${codeError}</p>
<p><input type="submit" value="提交"></p>
</form>
</body>
</html>
```

/category/listAll对应src/main/webapp/WEB-INF/category/listAll.jsp，其代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!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>列出所有Category</title>
</head>
<body>
<table border="1" width="500">
      <tr>
            <td>code</td>
            <td>name</td>
      </tr>
      <c:forEach items="${categories }" var="category">
      <tr>
            <td>${category.name }</td>
            <td>${category.code }</td>
      </tr>
      </c:forEach>
</table>
</body>
</html>
```

上述代码使用JSTL(http://baike.baidu.com/view/73527.htm?fr=aladdin)来控制页面显示，出于演示的目的，上述代码也没有对页面进行任何美化。最终程序执行的效果如下：

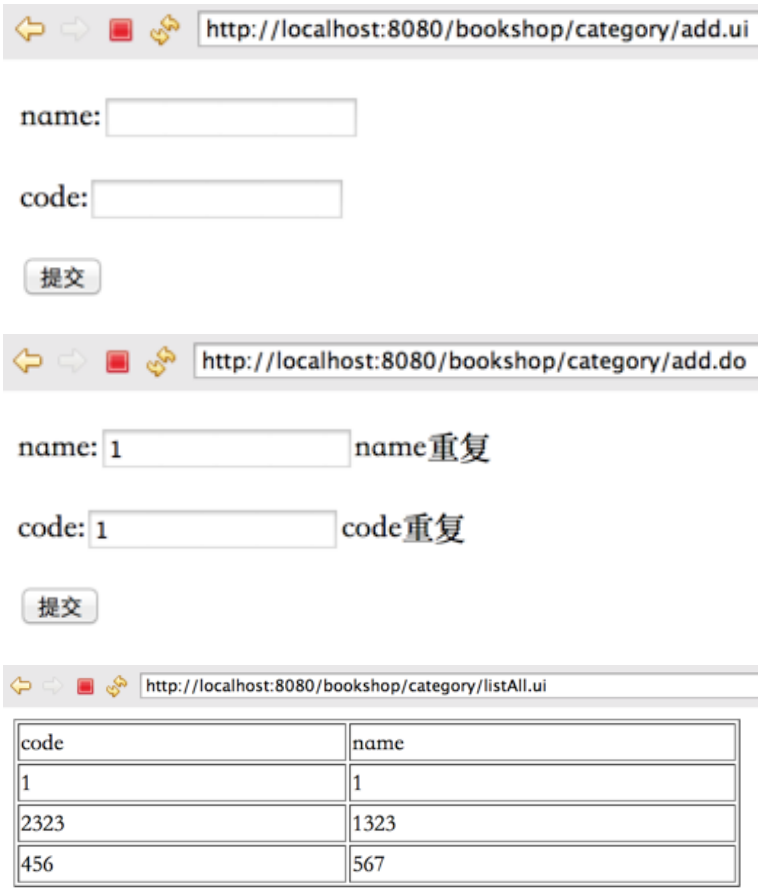


图 13 CRUD Web程序执行效果

注意：要恢复工程到本小节可以在命令行下进入工程根目录，然后键入指令：
git checkout v7
或者用Eclipse切换到Tag v7

2.6 总结

现在我们来总结一下，用Spring+Hibernate构建Web程序的一般步骤：

(1)需求分析，定义响应客户请求的URL地址(例如/category/add.ui)，定义Controller类，定义业务类接口(例如CategoryCRUD)，定义数据库及对应的领域类(Domain)、定义View(例如/category/addCategory)。

(2)编写业务接口的测试用例，例如CategoryCRUDHibernateTest

(3)编写业务接口的实现类，例如CategoryCRUDHibernate

(4)测试业务类

(5)编写Controller类的测试用例，例如CategoryControllerTest

(6)编写Controller类，例如CategoryController

(7)测试Controller类

(8)编写View

(9)测试

在实际项目中，流程(2)-(4)、(5)-(7)以及(8)可以并行的进行。也就是在流程(1)结束之后[2]，(2)-(4)、(5)-(7)以及(8)分别交给3个专门的团队(或者人)去完成。最终在流程(9)将Model、View和Controller整合起来。在实际项目中，流程(2)、(5)也可由单独的测试团队完成。

读者一定有疑问，如果流程(2)-(4)都没有完成，流程(5)-(7)怎么能够做呢？同理，流程(8)不是依赖于之前的流程吗？其实不然。流程(5)-(7)只依赖于业务类的接口，在实现过程中，程序员可以自定义一个简单的业务接口实现(例如定义一个实现了CategoryCRUD接口的类CategoryCRUD4Test)，用这个业务类来调试Controller。这个业务接口的实现类(CategoryCRUD4Test)仅仅为了测试，不仅可以不访问数据库，而且可以将逻辑写死在代码中。同理流程(8)在测试时也可以构造一些用于测试的数据。

上述流程结合Spring+Hibernate的程序架构，能够很好的保证Web应用的产品质量，特别是几乎每一个模块都会进行单元测试，模块之间耦合度低，便于今后的修改和扩展。但这种开发模式的副作用也显而易见。如果整个程序是由1-2个人来完成的，那么这1-2个人的工作强度可想而知。故此上述模式是一种适合工业生产的开发模式，而不是手工作坊(1-2个人)的开发模式。如果团队有4人，那么可以由1人编写业务类、1人编写Controller类、1人编写View、1人负责测试。此时上述模式才能真正体现威力。

3. 改进View

Javascript、Ajax和ExtJS都是View端的技术。本章我们将利用这些技术改进View。

3.1 用Javascript改进View

在前面的例子中/category/add.do在Controller端对浏览器提交的数据进行有效性验证。这种方式的数据有效性验证也称为服务器端的有效性验证。服务器端的数据有效性验证的优点在于数据有效性的验证无法被客户端略过，能够确保数据的正确性。但这种方式也存在弊端，包括：

(1)增加了服务器端的负担，使得Controller变得复杂。

(2)用户需要先见数据提交到服务端，然后才能获知数据是否正确录入，用户界面不够友好。

(3)一旦数据出错，多数情况下需要重建用户的输入界面(包含用户录入的数据)，因而出错处理比较复杂。

因此另一种可选的做法是在客户端对数据进行有效性验证。由于客户端是浏览器，故此使用Javascript在浏览器中对数据进行验证[3]。我们使用JQuery来简化这个过程，对于JQuery不熟悉的可以参考此教程http://www.w3school.com.cn/jquery/index.asp。

我们将jquery.js文件放置在src/main/webapp/resources/scripts目录下，然后如下修改src/main/webapp/WEB-INF/views/category/addCategory.jsp：

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://
www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<script type="text/javascript" src="<%=request.getContextPath()%>/
resources/scripts/jquery.js"></script>
<script type="text/javascript">
```

[2] 设计时需要保证Model、Controller和View之间是低耦合的，并且各个类能够进行单元测试。

[3] 如果客户端是Android手机，那么就是在手机App中进行验证；如果客户端是PC程序，那么就在PC程序中进行验证。根据客户端的不同，验证所用的技术也有差异。

```
$(document).ready(function(){
	$("#categoryForm").submit(function(e){
		var name=$("#name").val();
		var code=$("#code").val();
		//检测code
		if(code.length<=0||code.length>10)
		{
			$("#codeError").html("code不能为空，且长度不能大于10");
			e.preventDefault();//禁止数据提交到服务器
		}
		//检测name
		if(name.length<=0||name.length>100)
		{
			$("#nameError").html("name不能为空，且长度不能大于100");
			e.preventDefault();//禁止数据提交到服务器
		}
	});
});
</script>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>添加Category</title>
</head>
<body>
<form id="categoryForm" action="add.do" method="POST">
<p>name:<input id="name" type="text" name="name" value="${name }"><span
id="nameError">${nameError}</span></p>
<p>code:<input id="code" type="text" name="code" value="${code }"><span
id="codeError">${codeError}</span></p>
<p><input type="submit" value="提交"></p>
</form>
</body>
</html>
```

程序执行结果如下：

图 14 Javascript数据验证

注意：要恢复工程到本小节可以在命令行下进入工程根目录，然后键入指令：

git checkout v8
或者用Eclipse切换到Tag v8

3.2 用Ajax改进View

由于在数据库中categories表的name和code字段是唯一约束的，这就意味着仍然需要将name和code的数据发送到服务器端才能判断name或者code是否重复。为了解决这个问题，可以使用Ajax。目前主流的Ajax实现都是与服务器端交互JSON数据，而在DemoController的实现中，我们已经演示了如何返回JSON数据。故此我们首先在CategoryController中增加检查name和code是否唯一，并且用JSON返回数据的函数。其代码如下：

```
	@RequestMapping(value="/category/isDuplicate.json")
	@ResponseBody
	public Map<String,Boolean> isDuplicate(String code,String name)
	{
		Map<String,Boolean> map=new HashMap<String,Boolean>();
		//检查不能存在相同的code
		if(categoryCRUD.getByCode(code)!=null)
		{
			map.put("codeError", true);
		}else
		{
			map.put("codeError", false);
		}
		//检查不能存在相同的name
		if(categoryCRUD.getByName(name)!=null)
		{
			map.put("nameError", true);
		}else
		{
			map.put("nameError", false);
		}
		return map;
	}
```

最后修改src/main/webapp/WEB-INF/views/category/addCategory.jsp，添加Ajax代码如下：

```
<script type="text/javascript">
$(document).ready(function(){
	$("#categoryForm").submit(function(e){

		var name=$("#name").val();
		var code=$("#code").val();
		var valid=true;
		//检测code
		if(code.length<=0||code.length>10)
		{
			$("#codeError").html("code不能为空，且长度不能大于10");
			e.preventDefault();//禁止数据提交到服务器
```

```
            }
            //检测name
            if(name.length<=0||name.length>100)
            {
                  $("#nameError").html("name不能为空，且长度不能大于100");
                  e.preventDefault();//禁止数据提交到服务器
            }
            if(valid)
            {
                  $.ajax({
                        url: "isDuplicate.json",
                        data: {
                              name: name,
                              code: code
                        },
                        async: false, //设置Ajax请求为同步方式
                        success: function( json ) {
                              if(json.codeError==true)
                              {
                                    $("#codeError").html("code重复(Ajax验
证)");
                                    e.preventDefault();//禁止数据提交到服务器
                              }
                              if(json.nameError==true)
                              {
                                    $("#nameError").html("name重复(Ajax验
证)");
                                    e.preventDefault();//禁止数据提交到服务器
                              }

                        }
                  });
            }
      });
});
</script>
```

执行情况如下：

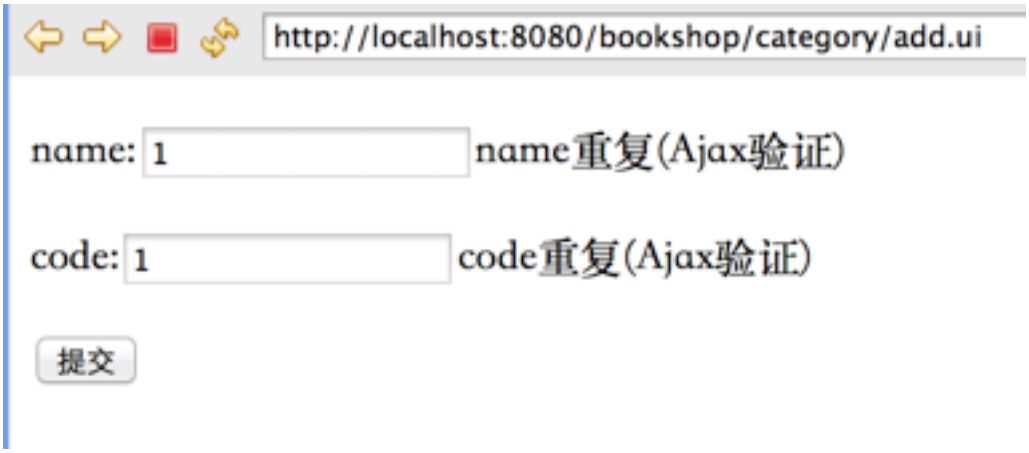


图 15 Ajax验证执行情况

通过使用Javascript和Ajax，我们将服务端的数据有效性验证，实现为客户端的(浏览器的)数据有效性验证。现在我们还有一个问题，那就是服务端的数据有效性验证代码是否可以就此删除呢？这应该视情况而定。因为客户端的数据有效性验证并不安全，特别是浏览器，有很多方式可以绕过。诸如身份证号码和信用卡号码这类的验证，不仅应该在客户端做(使交互界面友好)，而且要在服务器端做(确保数据安全)。

注意：要恢复工程到本小节可以在命令行下进入工程根目录，然后键入指令：
git checkout v9
或者用Eclipse切换到Tag v9

3.3 用ExtJS改进View

ExtJS是一个Javascript框架，主要用于构建复杂的Web界面。学习ExtJS需要掌握以下几个内容：

(1)有哪些界面组件
(2)如何对界面组件进行布局
(3)如何处理组件事件
(4)组件如何与服务端程序进行交互(通常使用Ajax，数据格式采用JSON)

要掌握这些内容，除了看书，官方的文档是最重要的资源。由于官网速度过慢，我在202.202.5.145上搭建了一个服务器，可以查看ExtJS的帮助文档。下面我们首先介绍一些Javascript的基础知识。

3.3.1 Javascript基础

尽管这部分叫Javascript基础但我们主要讲一讲可能会遇到的一些古怪符号和用法，毕竟Javascript的语法还是很简单，若不清楚可以参考其他资料。

(1)定义对象

如下语法可以定义一个对象：

```
var obj={
        name:"abc",
        age: 10
};
```

通过obj.name和obj.age就可以访问name和age属性

(2)定义函数

如下语法可以定义函数：

```
function add(a,b)
{
    return a+b;
}
```

或者

```
var add=function(a,b){
    return a+b;
}
```

上述定义完全等价，add(1,2)实现相同的效果。

(3)为对象添加方法

如下语法为定义的对象添加方法：

```
var obj={
            name:"abc",
            age: 10,
            add: function(a,b){
                    return a+b;
            }
};
```

通过obj.add(1,2)，调用obj对象的add方法。

(4)函数作为参数

实际上就是C的函数指针，例如：

```
Ext.onReady(function(){
    //do something
});
```

等价于

```
var f= function(){
    //do something
}
Ext.onReady(f);
```

或者

```
function f(){
    //do something
}
Ext.onReady(f);
```

(5)数组

如下代码定义数组：

```
var arr=[1,2,3];
```

数组的元素可以是任何值：

```
var obj={
            name:"abc",
            age: 10
};
var arr=[1,obj,3];
```

或者

```
var arr=[1, {
            name:"abc",
            age: 10
},3];
```

(6)Dom元素

HTML文档上的标签称为Dom元素。例如：

```
<div id="mydiv"></div>
```

Javascript可以获取Dom元素，并操纵它。对于有id属性的Dom元素，JQuery可以用如下方式访问到这个元素：

```
var mydi=$("#mydiv");
```

如果在ExtJS中可以用如下代码：

```
var mydiv=Ext.get("mydiv");
```

Ext.get返回的是一个Ext.dom.Element类型。通过查找文档，我们可以知道如何操纵这个类型。

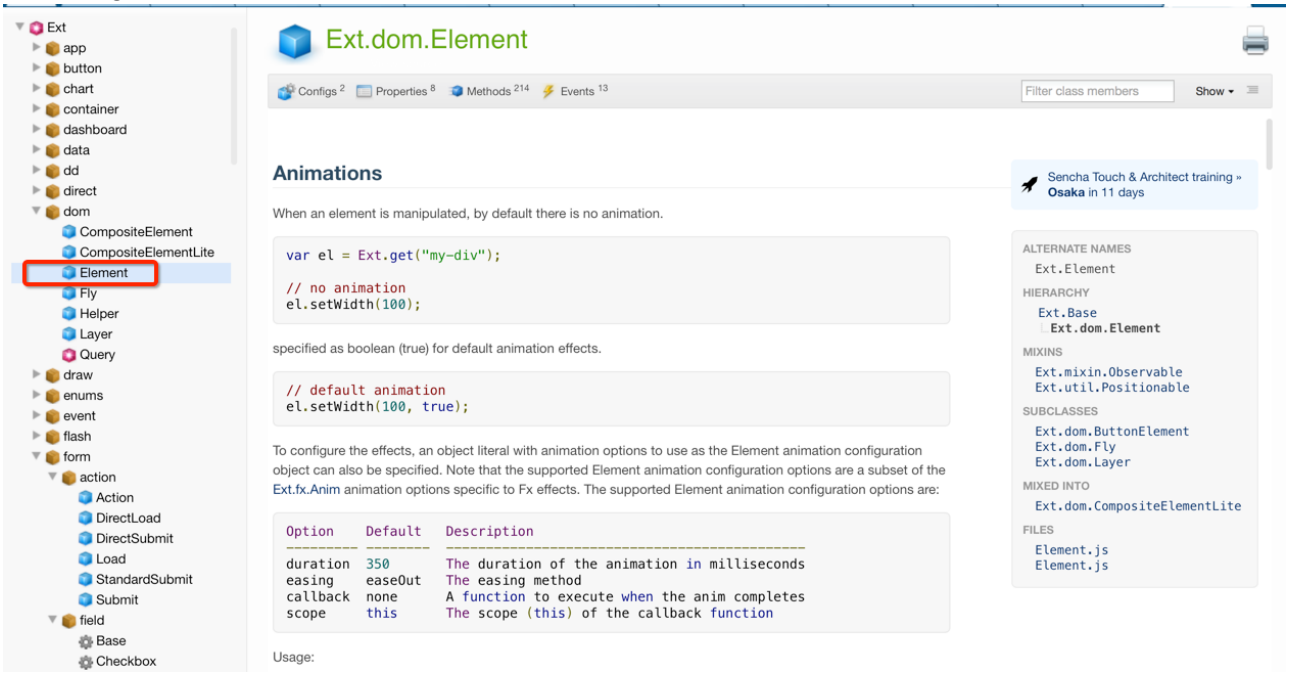


图 16 Ext.dom.Element的帮助文档

通过查找文档我们可以知道用setHtml方法可以为mydiv设置HTML内容。

3.3.2 改进添加Category

首先我们将ExtJS的全部文件放置在src/main/webapp/resources/scripts/extjs目录下。由于ExtJS非常大，Eclipse需要关闭Javascript文件的验证功能(示例已经关闭)，实际部署项目时还需要通过服务器配置加速下载[4]。为了使用ExtJS需要在所有View的<head>标签中添加：

```
<link rel="stylesheet" type="text/css"
      href="<%=request.getContextPath()%>/resources/scripts/extjs/
packages/ext-theme-neptune/build/resources/ext-theme-neptune-all.css">
<script
      src="<%=request.getContextPath()%>/resources/scripts/extjs/ext-
all.js"></script>
```

[4] 可以考虑配置Apache Http Server的代理转发

首先我们修改一下src/main/webapp/WEB-INF/views/category/addCategory.jsp，代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://
www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<link rel="stylesheet" type="text/css"
    href="<%=request.getContextPath()%>/resources/scripts/extjs/
packages/ext-theme-neptune/build/resources/ext-theme-neptune-all.css">
<script
    src="<%=request.getContextPath()%>/resources/scripts/extjs/ext-
all.js"></script>

<script type="text/javascript">
    //定义表单提交后处理服务器返回数据的函数
    var onSuccessOrFail = function(form, action) {
        var formPanel = Ext.getCmp('myFormPanel');
        formPanel.el.unmask();
        var result = action.result;
        if (result.success) {
            Ext.MessageBox.alert('成功', result.msg, function() {
                window.location.href = "listAll.ui";
            });

        } else {
            Ext.MessageBox.alert('失败', result.msg);
        }
    };
    //定义表单提交按键处理函数
    var submitHandler = function(btn) {
        var formPanel = Ext.getCmp('myFormPanel');
        formPanel.el.mask('正在提交数据请等待...', 'x-mask-loading');
        formPanel.getForm().submit({
            url : 'addCategory.do',
            success : onSuccessOrFail,
            failure : onSuccessOrFail
        });
    };
    //创建提交按键
    var submitBtn = Ext.create('Ext.button.Button', {
        text : '提交',
        handler : submitHandler,
    });
    //创建表单输入框（名称和编码）
```

```
	var fpItems = [ {
		fieldLabel : '名称:',
		name : 'name',
		allowBlank : false,
		blankText : '名称不能为空',
		maxLength : 100,
		maxLengthText : '名称不能超过100'
	}, {
		fieldLabel : '编码:',
		name : 'code',
		allowBlank : false,
		blankText : '编码不能为空',
		maxLength : 10,
		maxLengthText : '编码不能超过10'
	} ];
	//创建表单
	var fp = Ext.create('Ext.form.Panel', {
		id : "myFormPanel",
		width : 400,
		height : 240,
		title : '添加Category',
		frame : true,
		region : 'center',
		layout : 'anchor',
		bodyStyle : 'padding: 6px',
		labelWidth : 50,
		defaultType : 'textfield',
		defaults : {
			msgTarget : 'under',
			anchor : '-20'
		},
		items : fpItems,
		buttons : [ submitBtn ]
	});
	//页面加载完毕后显示表单
	Ext.onReady(function() {
		//表单显示到formDiv
		fp.render(Ext.get("formDiv"));
	});
</script>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>添加Category</title>
</head>
<body>
```

```
    <!-- formDiv位于页面正中（距离顶部100px），用于显示表单 -->
     <div id="formDiv" style="margin: 100px auto; width: 400px"></div>
</body>
</html>
```

由于采用Ajax向服务端提交数据，并且ExtJS的表单元素(Ext.form.Panel)内建了数据验证功能，所以很多工作都得到了简化。我们可以修改服务端程序。为了描述表单提交后的返回数据，我们定义FormResponse类：

```
package cn.edu.cqu.bookshop.json;

import java.util.HashMap;
import java.util.Map;

public class FormResponse {
	private boolean success=true;
	private String msg;
	private Map<String,String> errors=new HashMap<String,String>();
	public void addError(String field,String msg)
	{
		errors.put(field, msg);
		if(success)
			success=false;
	}
	public boolean hasError()
	{
		return !errors.isEmpty();
	}
	public boolean isSuccess() {
		return success;
	}
	public void setSuccess(boolean success) {
		this.success = success;
	}
	public String getMsg() {
		return msg;
	}
	public void setMsg(String msg) {
		this.msg = msg;
	}
	public Map<String, String> getErrors() {
		return errors;
	}
	public void setErrors(Map<String, String> errors) {
		this.errors = errors;
	}
```

```
}
```

修改CategoryController如下：

```
@RequestMapping(value = "/category/addCategory.do")
@ResponseBody
public FormResponse add(String name, String code) {
	FormResponse response = new FormResponse();
	try {
		if (categoryCRUD.getByCode(code) != null) {
			response.addError("code", "编码重复");
		}
		if (categoryCRUD.getByName(name) != null) {
			response.addError("name", "名称重复");
		}
		if (response.hasError()) {
			response.setMsg("数据异常,无法添加！");
		} else {
			Category category = new Category();
			category.setCode(code);
			category.setName(name);
			categoryCRUD.add(category);
			response.setMsg("添加成功！");

		}
	} catch (Exception e) {
		response.setMsg("系统错误，无法添加！");
		response.setSuccess(false);
		e.printStackTrace();
	}
	return response;
}
```

/category/addCategory.do若执行失败，其返回的json数据如下：

```
{"success":false,"msg":"系统错误，无法添加！","errors":{}}
```

或者

```
{"success":false,"msg":"数据异常,无法添加！","errors":{"code":"编码重复
","name":"名称重复"}}
```

如果成功，其返回数据如下：

```
{"success":true,"msg":"添加成功！","errors":{}}
```

此返回数据由处理，其定义如下：

```
var onSuccessOrFail = function(form, action) {
	var formPanel = Ext.getCmp('myFormPanel');
	formPanel.el.unmask();
	var result = action.result;
	if (result.success) {
```

```
                Ext.MessageBox.alert('成功', result.msg, function() {
                     window.location.href = "listAll.ui";
                });

           } else {
                Ext.MessageBox.alert('失败', result.msg);
           }
     };
```

ExtJS的表单会自动识别errors元素并显示服务端返回的错误信息。程序执行效果如下：

图 17 ExtJS执行效果

3.3.3 改进列出全部Category

此处我们使用ExtJS的Grid来显示数据。数据通过JSON进行加载，首先修改CategoryController，加入返回JSON数据的代码：

```
     @RequestMapping(value = "/category/categories.json")
     @ResponseBody
     public List<Category> categories()
     {
          List<Category> categories = categoryCRUD.getAll();
          return categories;
     }
```

然后修改src/main/webapp/WEB-INF/views/category/listAll.jsp，代码如下：

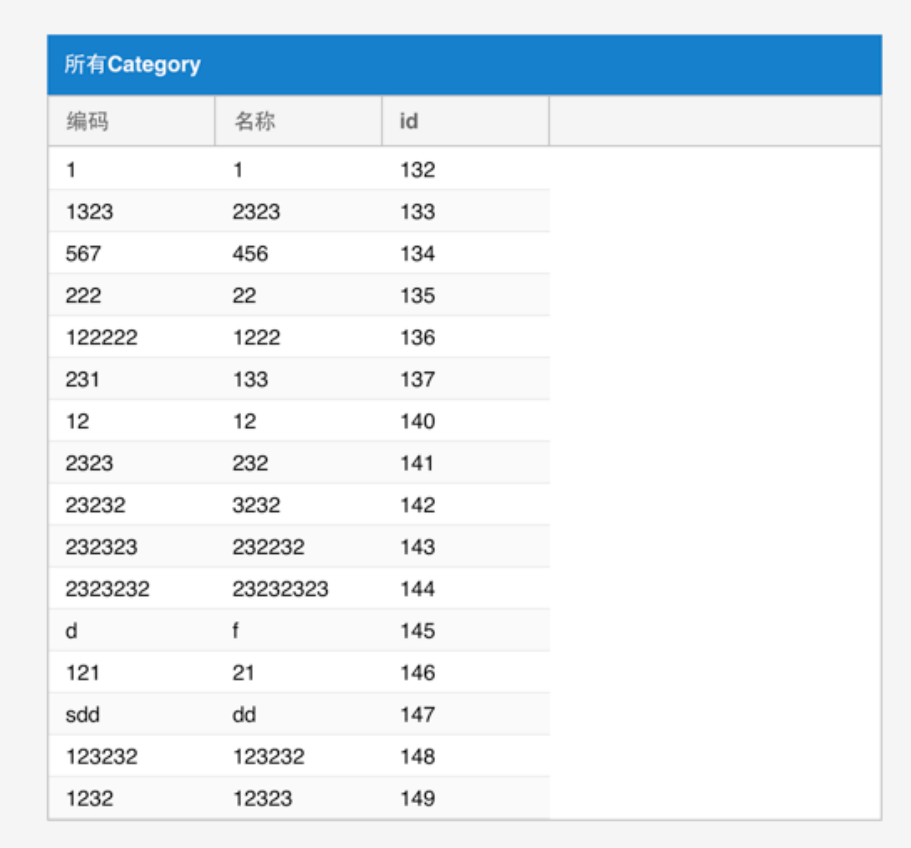
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
     pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
```

```
<!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">
<link rel="stylesheet" type="text/css"
      href="<%=request.getContextPath()%>/resources/scripts/extjs/
packages/ext-theme-neptune/build/resources/ext-theme-neptune-all.css">
<script
      src="<%=request.getContextPath()%>/resources/scripts/extjs/ext-
all.js"></script>
<script type="text/javascript">
    //定义JSON数据源
      var store = new Ext.data.JsonStore({
            storeId : 'myStore',
            proxy : {
                  type : 'ajax',
                  url : 'categories.json',
                  reader : {
                        type : 'json',
                        rootProperty : ''
                  }
            },
            fields : [ 'name', 'code', 'id' ]
      });
      Ext.onReady(function() {
            //加载JSON数据
            store.load(function(records, operation, success) {
                  if (success) {
                        //若加载成功创建grid显示数据
                        var grid = Ext.create('Ext.grid.Panel', {
                              title : '所有Category',
                              renderTo : Ext.get("gridDiv"),
                              autoHeight : true,
                              width : 500,
                              store : store,
                              selType : 'rowmodel',
                              singleSelect : true,
                              columns : [ {
                                    header : '编码',
                                    sortable : true,
                                    dataIndex : 'code'
                              }, {
                                    header : '名称',
                                    sortable : false,
```

```
                                dataIndex : 'name'
                            }, {
                                header : 'id',
                                sortable : false,
                                dataIndex : 'id'
                            } ]
                        });
                }else{
                        Ext.MessageBox.alert('失败', "数据加载失败");
                }
            });

    });
</script>
<title>列出所有Category</title>
</head>
<body>
    <!-- gridDiv位于页面正中（距离顶部50px），用于显示grid -->
    <div id="gridDiv" style="margin: 50px auto; width: 500px"></div>
</body>
</html>
```

程序效果如下：



所有Category

编码	名称	id
1	1	132
1323	2323	133
567	456	134
222	22	135
122222	1222	136
231	133	137
12	12	140
2323	232	141
23232	3232	142
232323	232232	143
2323232	23232323	144
d	f	145
121	21	146
sdd	dd	147
123232	123232	148
1232	12323	149

图 18 显示全部Category

Grid还有很多高级功能，包括分页显示、添加功能按键、编辑数据等，在此不再一一列举，具体做法可以参考其他资料。

注意：要恢复工程到本小节可以在命令行下进入工程根目录，然后键入指令：
git checkout v10
或者用Eclipse切换到Tag v10

3.3.4 几点讨论

读者应该已经发现，用了ExtJS之后程序的架构发生了少许变换，具体来说形成了如下图所示的结构：

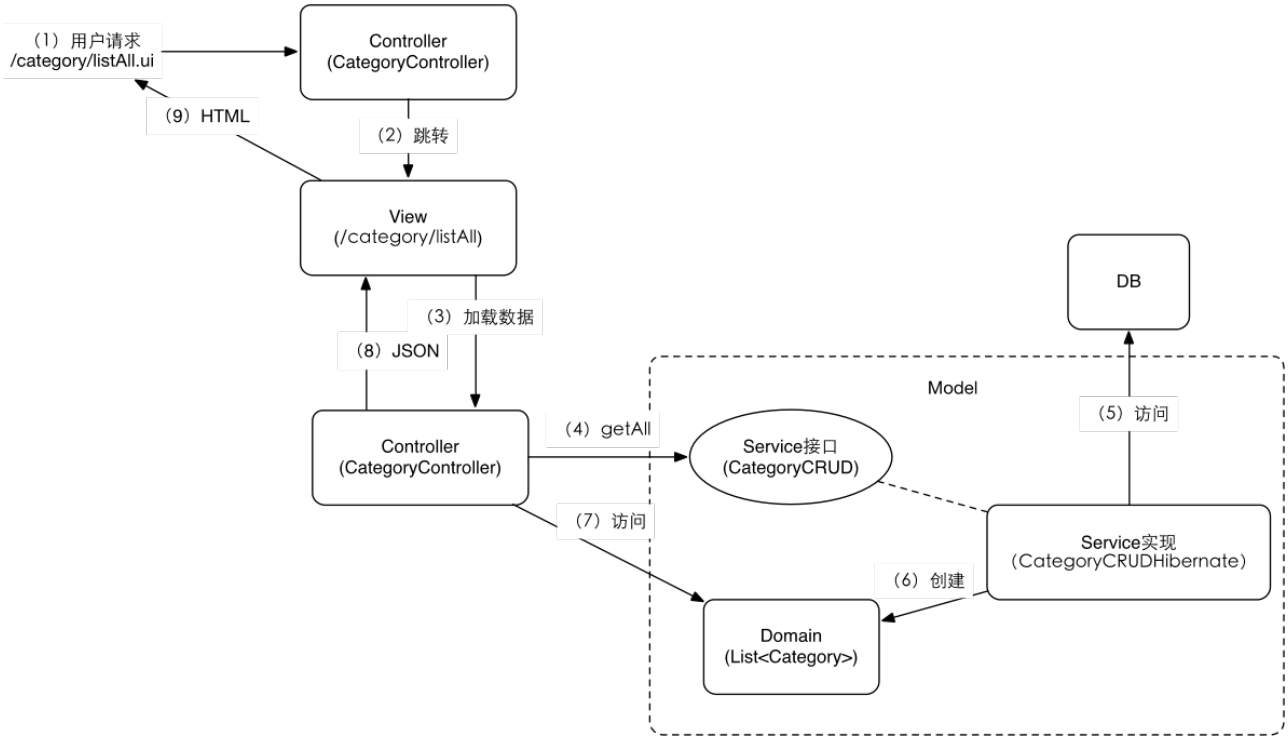


图 19 ExtJS程序结构

这个程序结构相对复杂，我们是否可以将其简化为如下的结构呢?

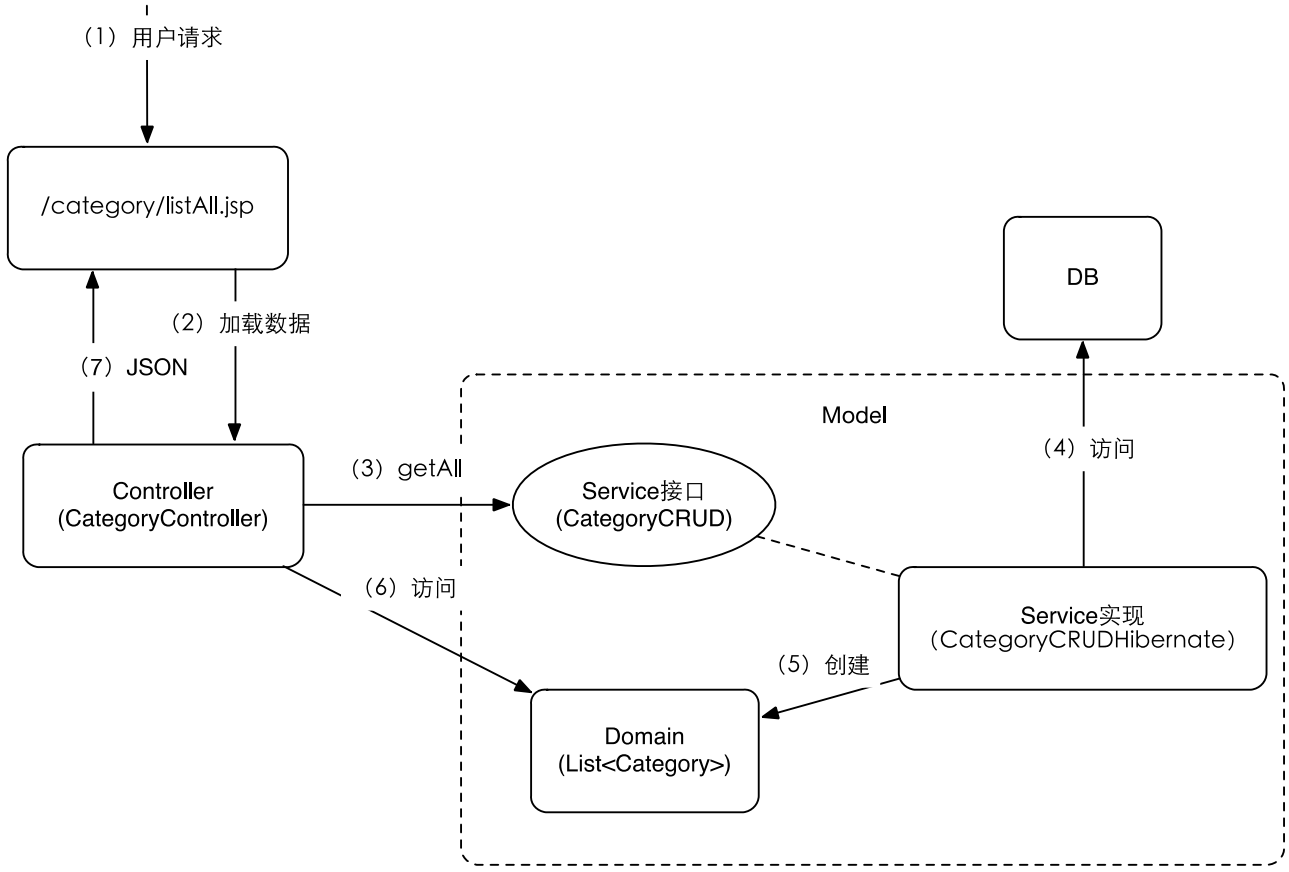


图 20 简化ExtJS的程序架构

换言之，也就是将/category/listAll.jsp前端的控制器取消掉。如果采用这种方式，则意味着View全部依赖客户端的Javascript代码来组织UI。故此我们甚至可以用HTML(而不是JSP，因为在这种情况下JSP的功能完全无用)来实现View。请直接查看工程源代码。

注意：要恢复工程到本小节可以在命令行下进入工程根目录，然后键入指令：
git checkout v11
或者用Eclipse切换到Tag v11