使用注解的方式，配置文件最少可以精简到三个，web.xml、applicationContext.xml和struts.xml。Hibernate可以完全交给Spring来管理，这样连hibernate.cfg.xml也省了。

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="SSH"
version="2.5">
  <display-name>ssh</display-name>
  <welcome-file-list>
    <welcome-file>addUser.jsp</welcome-file>
  </welcome-file-list>

  <!-- 配置Spring的监听器，用于初始化ApplicationContext对象 -->
  <listener>
    <listener-class>
         org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:applicationContext*.xml</param-value>
  </context-param>

  <!-- struts2 的配置 -->
  <filter>
    <filter-name>struts2</filter-name>
        <filter-class>
 org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
       </filter-class>
    <init-param>
      <param-name>filterConfig</param-name>
      <param-value>classpath:struts.xml</param-value>
    </init-param>
```

```
    <!-- 自动扫描action -->
    <init-param>
      <param-name>actionPackages</param-name>
      <param-value>com.ssh</param-value>
    </init-param>
  </filter>

  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
```

web.xml中包含了Spring和struts的基本配置，自动扫描Action的配置就是告诉tomcat，我要使用注解来配置struts。

applicationContext.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"
    xmlns:tx="http://www.springframework.org/schema/tx"
    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
                http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

    <!-- 自动扫描与装配bean -->
    <context:component-scan base-package="com.tgb.ssh">
</context:component-scan>

    <!-- dbcp配置 -->
    <bean id="dataSource"
class="org.apache.commons.dbcp2.BasicDataSource" destroy-
method="close">
```

```
        <property name="driverClassName">
            <value>com.mysql.jdbc.Driver</value>
        </property>
        <property name="url">
            <value>jdbc:mysql://127.0.0.1:3307/ssh</value>
        </property>
        <property name="username">
            <value>root</value>
        </property>
        <property name="password">
            <value>123456</value>
        </property>
    </bean>

    <bean id="sessionFactory"
class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
        <property name="dataSource">
            <ref local="dataSource" />
        </property>
        <property name="hibernateProperties">
            <props>
                <!--配置Hibernate的方言-->
                <prop key="hibernate.dialect">
                 org.hibernate.dialect.MySQLDialect
                </prop>
                <prop key="hibernate.hbm2ddl.auto">update</prop>

                <!--格式化输出sql语句-->
                <prop key="hibernate.show_sql">true</prop>
                <prop key="hibernate.format_sql">true</prop>
                <prop key="hibernate.use_sql_comments">false</prop>
            </props>
        </property>

        <!--自动扫描实体 -->
        <property name="packagesToScan"  value="com.tgb.ssh.model" />
    </bean>

    <!-- 用注解来实现事务管理 -->
```

```
    <bean id="txManager"
class="org.springframework.orm.hibernate4.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory">
</property>
    </bean>
    <tx:annotation-driven transaction-manager="txManager"/>

</beans>
```

applicationContext.xml里配置了数据库连接的基本信息（对hibernate的管理），还有对所有bean的自动装配管理和事务的管理。

struts.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
"http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

    <!-- 开启使用开发模式，详细错误提示 -->
    <constant name="struts.devMode" value="true" />
    <!-- 将对象交给spring管理 -->
    <constant name="struts.objectFactory" value="spring" />
    <!-- 指定资源编码类型 -->
    <constant name="struts.i18n.encoding" value="UTF-8" />
    <!-- 指定每次请求到达，重新加载资源文件 -->
    <constant name="struts.i18n.reload" value="false" />
    <!-- 指定每次配置文件更改后，自动重新加载 -->
    <constant name="struts.configuration.xml.reload" value="false" />
    <!-- 默认后缀名 -->
    <constant name="struts.action.extension" value="action," />

</struts>
```

struts.xml里配置了一些struts的基本参数，并告诉容器用Spring来管理自己。

到这里一个基本的SSH的配置就算完成了，配置很简单，而且每一项配置都有说明，相信理解上不会有什么问题。

基础的配置就这么多，下面就是我们的注解发挥作用的时候了。

userAdd.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>添加用户</title>
  </head>

  <body>
      <form method="post" action="addUser">
        用户名：<input type="text" name="user.name"><br>
        密码：<input type="password" name="user.password"><br>
        <input type="submit" value="登录"/>
    </form>
  </body>
</html>
```

用户添加页面，将用户信息提交给UserAction。

UserAction

```
package com.tgb.ssh.action;

import javax.annotation.Resource;

import org.apache.struts2.convention.annotation.Action;
import org.apache.struts2.convention.annotation.Result;
import org.apache.struts2.convention.annotation.Results;

import com.opensymphony.xwork2.ActionSupport;
import com.tgb.ssh.model.User;
import com.tgb.ssh.service.UserManager;

@Results( { @Result(name="success",location="/success.jsp"),
        @Result(name="failure",location="/failure.jsp") })
public class UserAction extends ActionSupport {
    @Resource
```

```
    private UserManager userManager;
    private User user;

    @Action(value="addUser")
    public String addUser() {
        try {
            userManager.addUser(user);
        } catch (Exception e) {
            e.printStackTrace();
            return "failure";
        }
        return "success";

    }

    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }

}
```

UserAction通过注解配置Action的名字和返回的页面，通过@Resource活动Spring注入的UserManager对象，然后进行相应的操作。

Action里还有@Namespace、@InterceptorRef等很多注解可以用，根据自己需要选择吧。

UserManager

```
package com.tgb.ssh.service;

import javax.annotation.Resource;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.tgb.ssh.dao.UserDao;
import com.tgb.ssh.model.User;
```

```
@Service
@Transactional
public class UserManager {
    @Resource
    UserDao userDao;

    public void addUser(User user) {
        userDao.addUser(user);
    }

}
```

UserManager通过@Service自动装配到Spring的容器，为其他组件提供服务；
通过@Transactional进行事务的管理；通过@Resource注入UserDao对象。

UserDao

```
package com.tgb.ssh.dao;

import javax.annotation.Resource;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.springframework.orm.hibernate4.HibernateTemplate;
import org.springframework.stereotype.Repository;

import com.tgb.ssh.model.User;

@Repository
public class UserDao {
    @Resource(name="sessionFactory")
    private SessionFactory sessionFactory;

    public void addUser(User user ) {
        Session session = sessionFactory.getCurrentSession();
        session.save(user);
    }

}
```

UserDao通过@Repository自动装配到Spring的容器，通过@Resource获得Sessionfactory，将
User对象持久化。

User

```
package com.tgb.ssh.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity(name="t_user")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    private String password;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
```

```
        this.password = password;
    }
}
```

User通过@Entity将实体类映射到数据库，生成t_user表，通过@Id定义表的Id，通过@GenerateValue定义Id的生成策略。

好了，到此为止，基于注解的SSH就算是搭建完成了。
基础的搭建已经使注解简洁的优势初现端倪，随着开发的进行，代码不断地增加，其简洁的风格相比传统配置文件的方式会更加明显。

因为如果采用配置文件的方式，每增加一个Action都需要在struts.xml和applicationContext.xml文件增加一段代码；每多一个实体，也需要多一个*.hbm.xml文件。配置文件泛滥是一件让人头疼的事情。