



Web应用构建技术

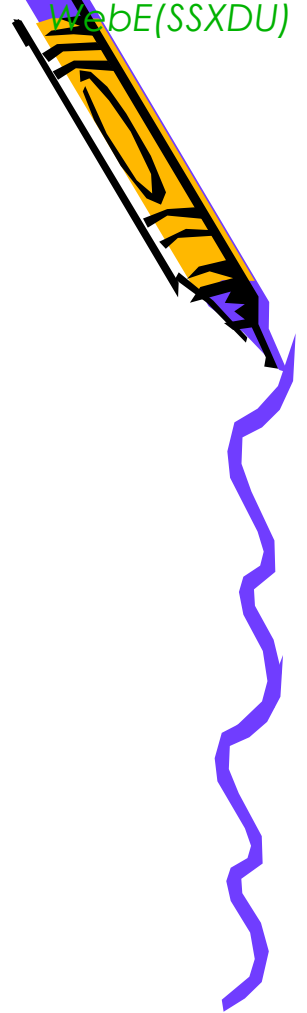
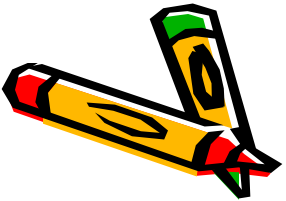
Backend – Spring, Hibernate

Qiuyan Huo 霍秋艳
Software Engineering Institute
qyhuo@mail.xidian.edu.cn



What is SSH?

- Spring, Struts, Hibernate



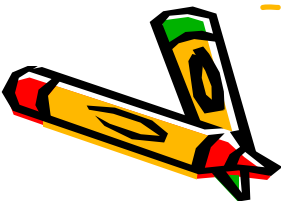
WebE(SSXDU)

Spring

- Open Source
- Open ecosystem
- Java-based应用平台
- 简化Java 企业应用
- 三层次
 - 核心：IoC容器、AOP
 - 组件：事务处理、Web MVC、JDBC、O/R映射、远程调用
 - 应用：Spring DM、Spring FLEX、ACEGI等等

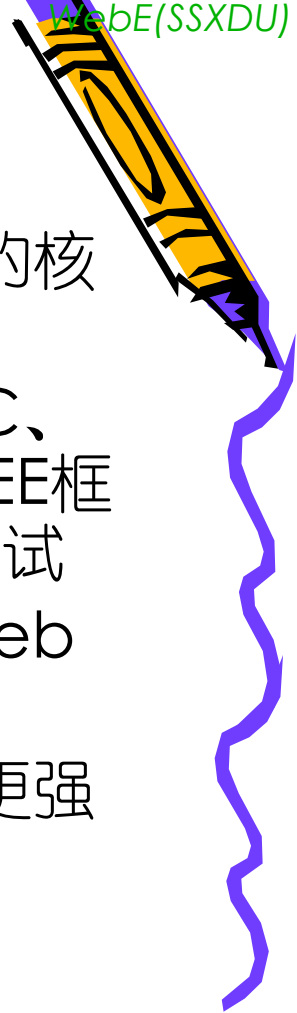
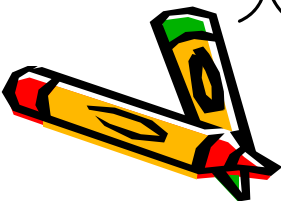
For the developer, to the developer and by the developer.

-- Rod Johnson



Spring

- Spring的核心理念是“不重复发明轮子”，而它的核心技术是依赖注入容器（IoC）
- Spring还包括：AOP、数据访问抽象、简化JDBC、事务管理、Web MVC、简化JNDI JTA和其他J2EE框架API、轻量级的远程访问、JMS、JMX、综合测试
- Spring可以很好的整合其他框架(持久化框架、Web框架、AOP框架等)
- Spring旨在让这些框架很好的集成在一起，提供更强大的服务



Spring - Projects



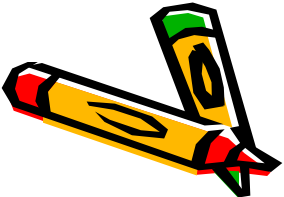
- Spring Boot

- Takes an opinionated view of building Spring applications and gets you up and running as quickly as possible.



- Spring Framework

- Provides core support for dependency injection, transaction management, web apps, data access, messaging and more.



Spring - Projects



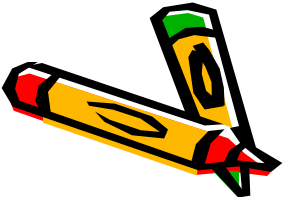
- Spring XD

- Simplifies the development of big data applications by addressing ingestion, analytics, batch jobs and data export.



- Spring Data

- Provides a consistent approach to data access – relational, non-relational, map-reduce, and beyond.



Spring - Projects



- Spring Integration

- Supports the well-known Enterprise Integration Patterns via lightweight messaging and declarative adapters.



- Spring Batch

- Simplifies and optimizes the work of processing high-volume batch operations.



- Spring Security

- Protects your application with comprehensive and extensible authentication and authorization support.



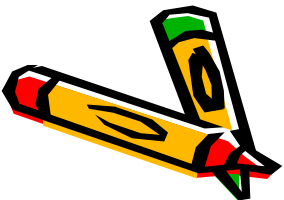
Spring - Projects



- Spring HATEOAS
 - Simplifies creating REST representations that follow the HATEOAS principle.



- Spring Social
 - Easily connects your applications with third-party APIs such as Facebook, Twitter, LinkedIn, and more.



Spring - Projects



- Spring AMQP

- Applies core Spring concepts to the development of AMQP-based messaging solutions.



- Spring Mobile

- Simplifies the development of mobile web apps through device detection and progressive rendering options.



- Spring for Android

- Provides key Spring components for use in developing Android applications.



Spring - Projects

WebE(SSXDU)



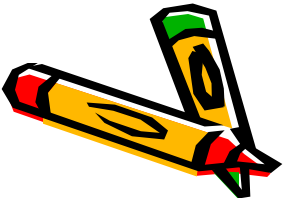
- Spring Web Flow

- Supports building web applications with controlled navigation such as checking in for a flight or applying for a loan.

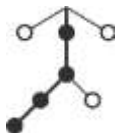


- Spring Web Services

- Facilitates the development of contract-first SOAP web services.



Spring - Projects



- Spring LDAP

- Simplifies the development of applications using LDAP using Spring's familiar template-based approach.



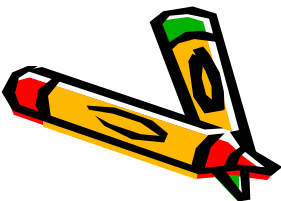
- Grails

- Builds on Spring to provide a full-stack environment for creating web applications using the Groovy language.



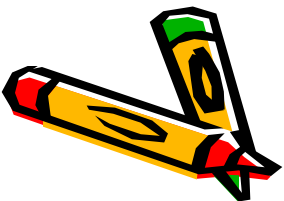
- Groovy

- Brings high-productivity language features to the JVM including support for static and dynamic programming, scripting, and domain-specific languages.



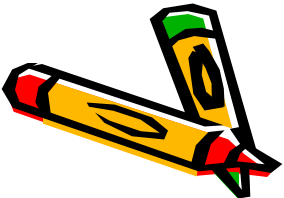
Spring - Projects

- Spring Scala
 - Brings the power and expressiveness of Scala together with the productivity and deep ecosystem of Spring.
- Spring Roo
 - Makes it fast and easy to build full Java applications in minutes.



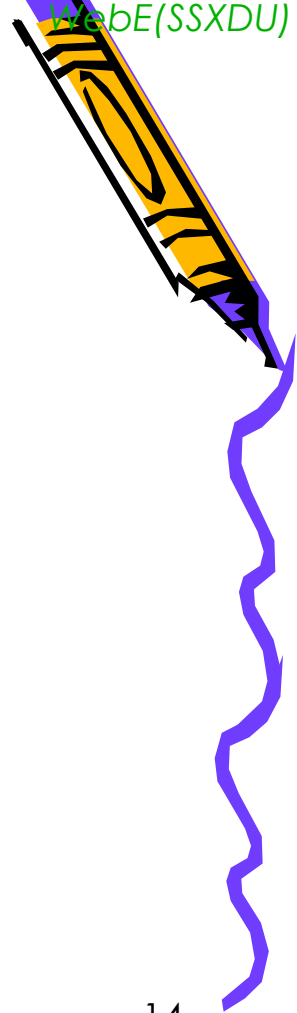
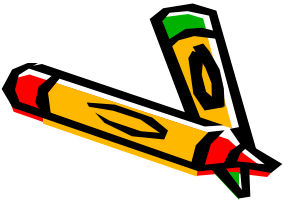
Spring - Projects

- Spring Blazeds Integration
 - Provides first-class support for using Adobe BlazeDS in Spring-based apps with Adobe Flex front-end clients.
- Spring Loaded
 - Boosts development productivity by reloading class file changes—as you make them—within your app's JVM.
- Spring Shell
 - Provides a powerful foundation for building command-line apps using a Spring-based programming model.
- Rest Shell
 - Makes writing and testing RESTful applications easier with CLI-based resource discovery and interaction.



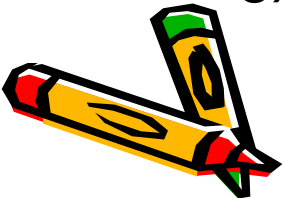
Spring Projects

SPRING DATA



Spring Projects - Spring Data

- Makes it easy to use new data access technologies, such as non-relational databases, map-reduce frameworks, and cloud based data services.
- Also provides improved support for relational database technologies.
- An umbrella project
 - contains many subprojects that are specific to a given database.
- Developed by working together with many of the companies and developers that are behind these exciting technologies.



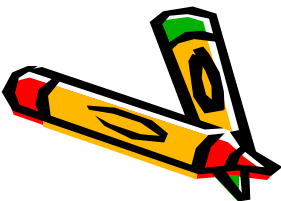
Spring Data - Main Projects

- Spring Data JPA
 - Makes it easy to implement JPA-based repositories.
- Spring Data MongoDB
 - Spring based, object-document support and repositories for MongoDB.
- Spring Data Neo4j
 - Spring based, object-graph support and repositories for Neo4j.
- Spring Data Redis
 - Provides easy configuration and access to Redis from Spring applications.



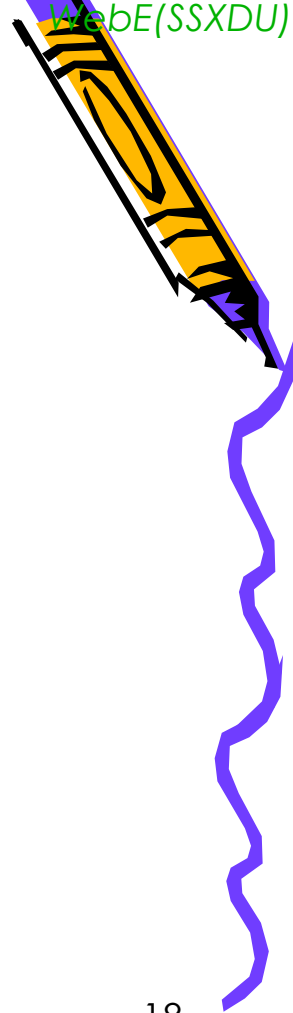
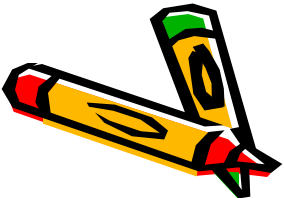
Spring Data - Main Projects

- Spring for Hadoop
 - Simplifies Apache Hadoop by providing a unified configuration model and easy to use APIs for using HDFS, MapReduce, Pig, and Hive.
- Spring Data GemFire
 - Provides easy configuration and access to GemFire from Spring applications.
- Spring Data REST
 - Exports Spring Data repositories as hypermedia-driven RESTful resources.
- Spring Data JDBC Extensions
 - Provides extensions to the JDBC support provided in the Spring Framework.



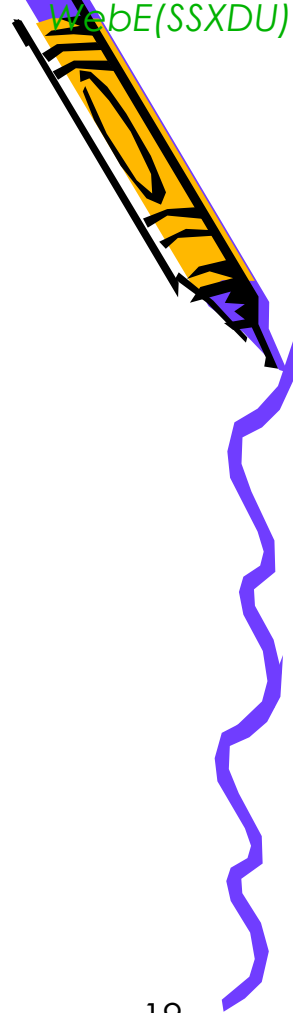
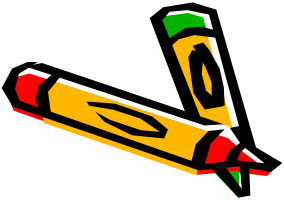
Spring Data – Community Projects

- Spring Data Solr
 - Spring Data module for Apache Solr.
- Spring Data Couchbase
 - Spring Data module for Couchbase.
- Spring Data Elasticsearch
 - Spring Data module for Elasticsearch.
- Spring Data Cassandra
 - Spring Data module for Cassandra.
- Spring Data DynamoDB
 - Spring Data module for DynamoDB.



Spring Projects

SPRING FRAMEWORK



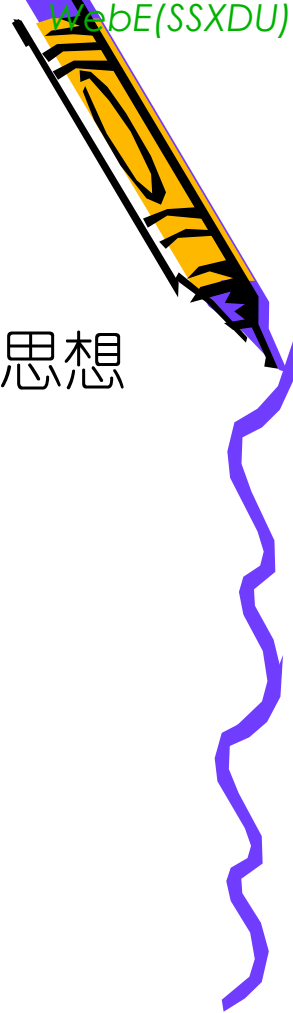
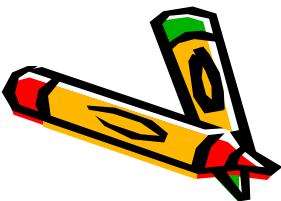
Spring和Web框架的整合

- Struts: Struts在Web MVC这个领域占据了很大一部分份额，但随着时间的推移，它的份额一直在下降。
- WebWork: WebWork组织对WebWork与Spring的整合有很大的兴趣，这极大的推动了WebWork的发展。
- Spring MVC: Spring自己的Web MVC框架，可以使用Spring的高特性的功能，这是其他框架不具备的。
- Tapestry: Apache Jakarta组织开发的一个面向构件的框架。
- JSF: Spring可以很好的整合JSF，JSP被称为“named beans”，有了Spring，它不需要实现自己的中间层服务。



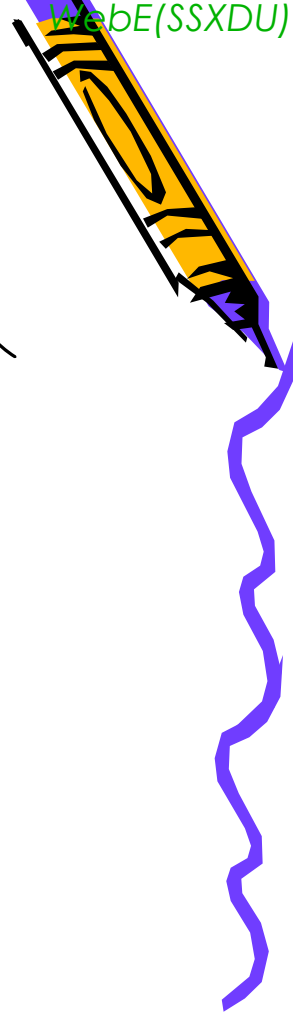
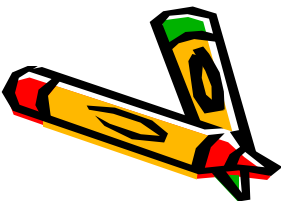
Spring MVC

- Spring MVC是一个非常灵活的框架。
- 通过配置文件和代码的注解可以很好的体现MVC思想。
 -
- Spring MVC开发示例→



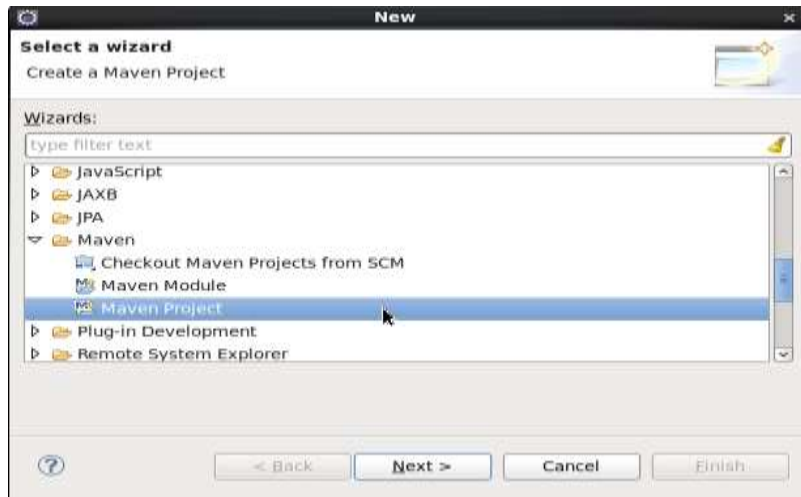
Maven

- 使用Maven来管理项目的构建。
- 下面我们将介绍使用Eclipse的maven来构建一个SpringMVC项目。



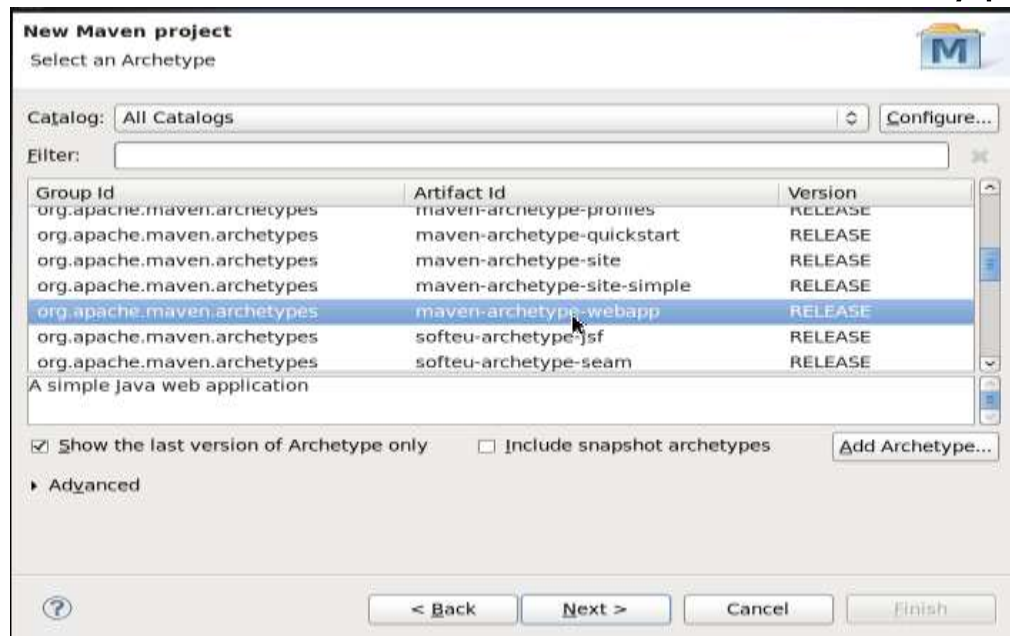
Start

- 首先确保Eclipse已经安装Maven。
- 选择File->new->other，在new窗口中选择Maven->Maven Project点击next。



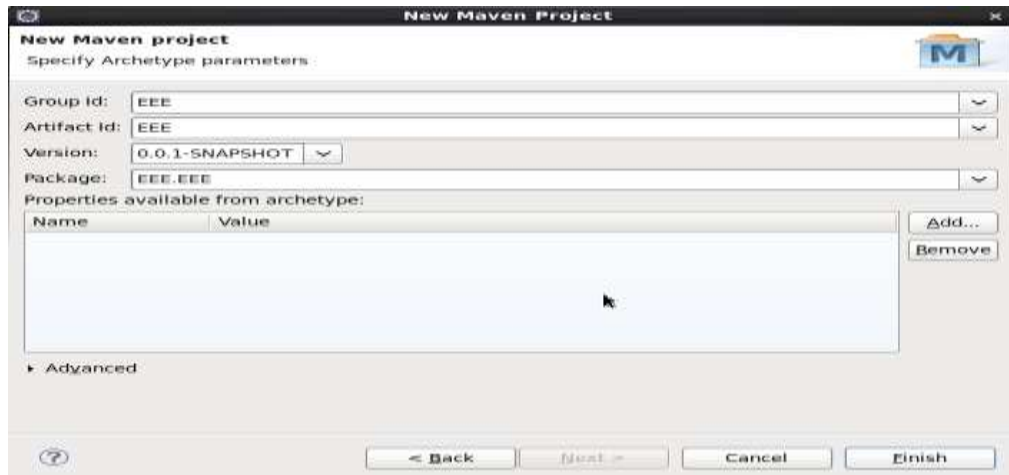
Process

- 选择项目空间路径
- 选择项目类型在Artifact Id中选择Maven-archetype-webapp



Process

- 输入Group ID和Artifact ID, 以及package
- Group ID一般写大项目名称, Artifact ID是子项目名称。
- 例如Spring的web包, Group ID:
org.springframework, artifact ID:spring-web
- Package是默认给你建一个包不写也可以。



New Maven Project

Specify Archetype parameters

Group Id: EEE

Artifact Id: EEE

Version: 0.0.1-SNAPSHOT

Package: EEE.EEE

Properties available from archetype:

Name	Value
------	-------

Advanced

< Back Next > Cancel Finish

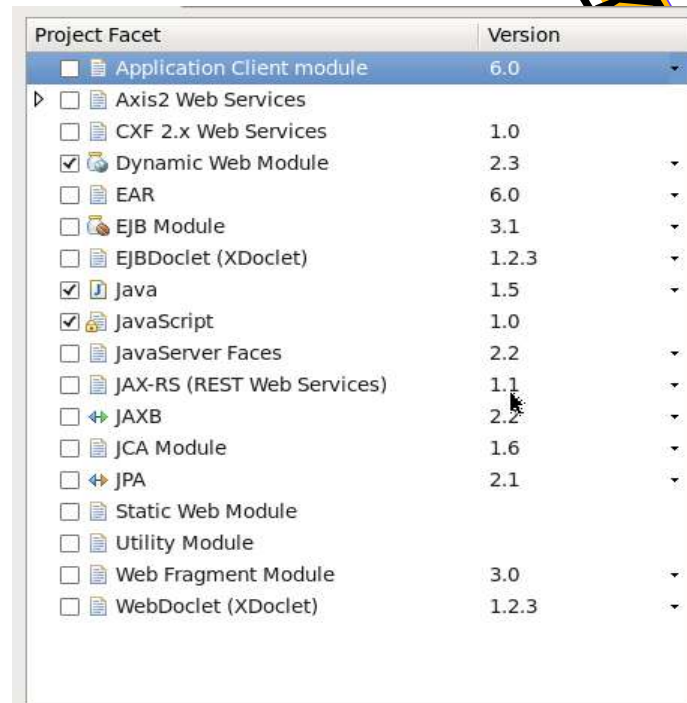
Process

- 建立好的项目的结构如右图：
- 接下来就要进行配置了。
- 添加source文件夹。
- 主要添加src/main/java,src/test/java,src/test/resources三个文件夹。
- 右键项目根目录点击New->Source Folder
- 建立这三个文件夹。



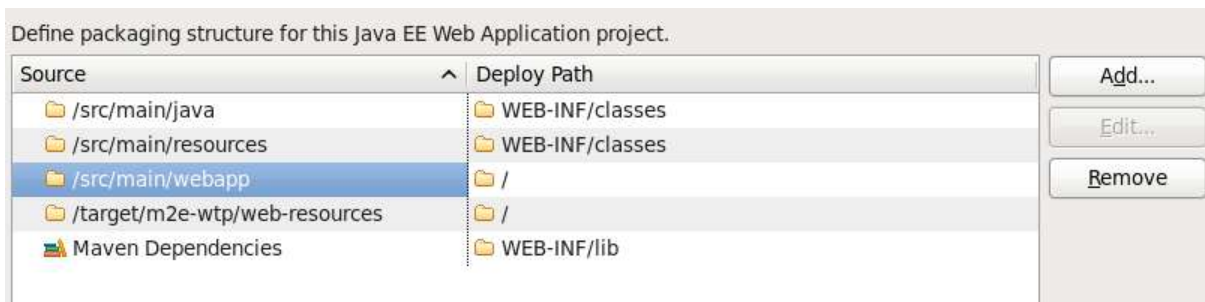
Process

- 把项目变成Dynamic Web项目
- 右键项目，选择Project Facets,
- 点击Convert to faceted from
- 配置Project Facets，更改Dynamic Web Model的version为2.5 (java6) (3.0为java7)
- 可能需要更改的地方：
- Java Compiler设置
- Compiler compliance level 为1.6



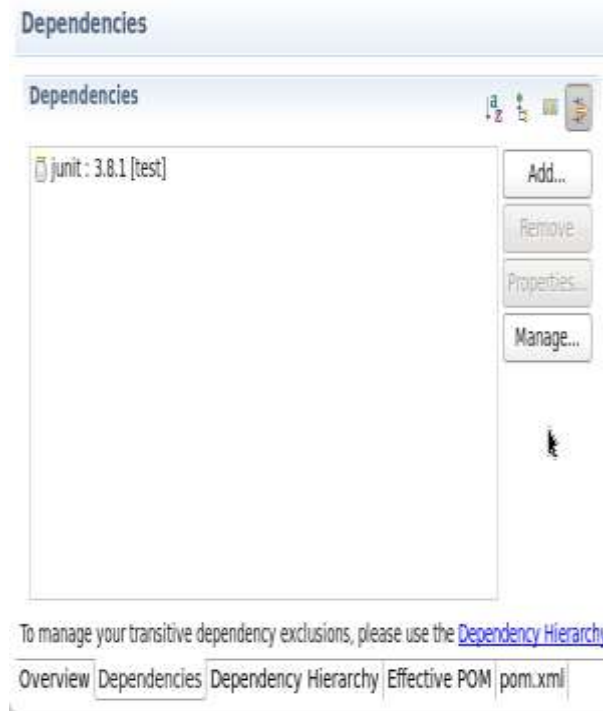
Process

- 设置部署程序集
- 右键项目打开Property窗口，左侧列表出现一个Deployment Assembly点击进去
- 设置Maven的jar包发布到lib下
- Add->Java Build Path Entries->Maven Dependencies->finish



Process

- 在pom.xml中添加需要的包
- 格式如下即可：
- `<dependency>`
- `<groupId>junit</groupId>`
- `<artifactId>junit</artifactId>`
- `<version>3.8.1</version>`
- `<scope>test</scope>`
- `</dependency>`
- 可在右图所示的pom.xml视图中点击Add进行添加搜索。
- 也可以去Maven提供找包的网站去找。



Finish

- 最后右键项目，选择Run As->Run on Server
- 添加tomcat7作为Server即可。
- 接下来需要修改和添加一些配置文件。
- 再接下来就是编写页面和对应的controller，和逻辑也业务层了。
- 页面在webapp下面建立
- 而controller和逻辑业务层java代码需要在之前新建的src/main/java中编写。



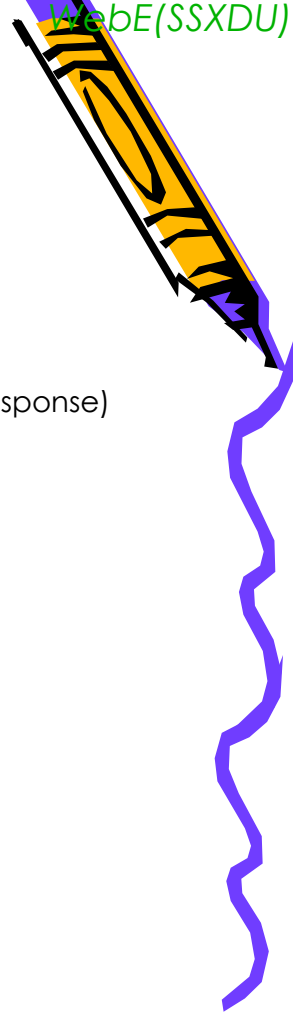
配置文件

- Web.xml的配置可以网上搜索具体的配置信息以及含义，然后进行自己的配置。
- 无外乎就是servlet,servlet-mapping,listener,filter,filter-mapping以及一些session
- 登陆首页错误页面等等的配置
- 进行spring-servlet.xml(名字可以在web.xml中自己定义)的配置，扫描包，上传文件的解析视图，json等的配置都可以写到这
- applicationContext.xml的配置，配置一些资源的映射与添加其他的配置文件信息如hibernate等。



Spring MVC开发示例——controller

```
@Component
public SimpleController
{
    @RequestMapping (value="/welcome",method={RequestMethod.GET})
    public ModelAndView handleRequestInternal(HttpServletRequest request,HttpServletResponse response)
    throws Exception
    {
        //创建一个视图welcome, 它是一个网页页面
        ModelAndView mav = new ModelAndView("welcome");
        // 向这个视图添加属性, 可以在页面中捕获到
        mav.addObject("date", new Date());
        return mav;
    }
}
```



Spring MVC开发示例——controller

- 在这段代码中：
- `@Component`注解表示这是一个Spring构件，通过Http发过来的URI被DispatcherServlet分配到这里。
- `@RequestMapping (value="/welcome", method={RequestMethod.GET})` 注解表示handleRequestInternal函数相当于一个model，完成URI为/welcome的Http请求的相应，而且这个请求必须以GET方式提交。
- 在handleRequestInternal中，ModelAndView对应了model函数结束之后讲好返回的页面，它的构造函数的参数表示不包含后缀的页面。
- 当用户发送了该Http请求之后，函数执行完成后会跳转到welcome页面。



Spring MVC开发示例--DispatcherServlet

- 配置DispatcherServlet：在web.xml文件中配置如下servlet（/*表示所有的请求都会通过DispatcherServlet分配）

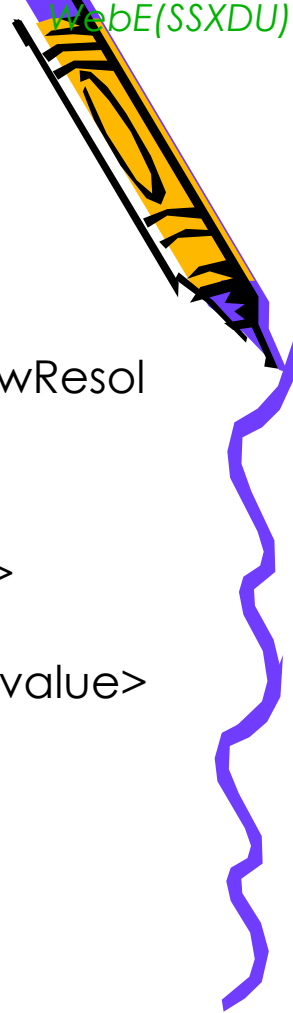
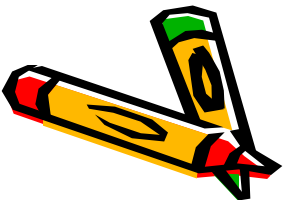
```
<servlet>
    <servlet-name>sample</servlet-name>
    <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servl
et-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>sample</servlet-name>
    <url-pattern>/*</url-pattern>
</servlet-mapping>
```



Spring MVC开发示例--application.xml

```
<beans>
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResol
ver">
    <property name="prefix"><value>/WEB-
INF/jsp</value></property>
    <property name="suffix"><value>.jsp</value></property>
    <property name="viewClass">
<value>org.springframework.web.servlet.view.JstlView</value>
</property>
</bean>
</beans>
```

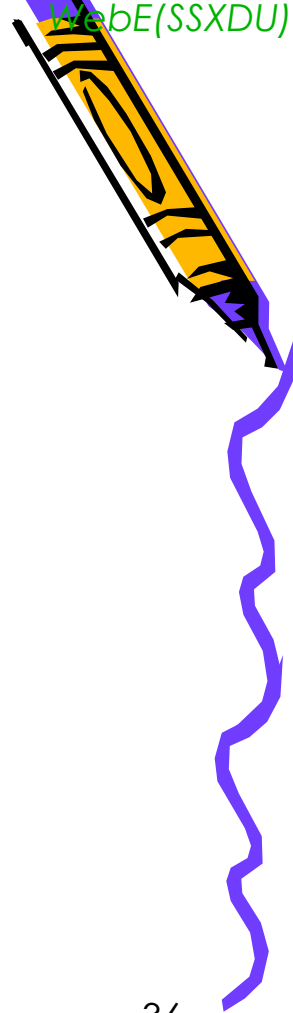
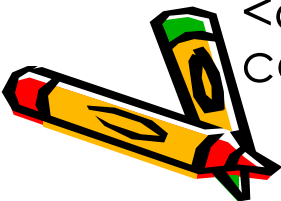
以上配置的bean, 表示了页面所在的位置是/WEB-INF/jsp, 页面的后缀是.jsp, 即在ModelAndView中的构造函数的参数不需要包含这些信息。



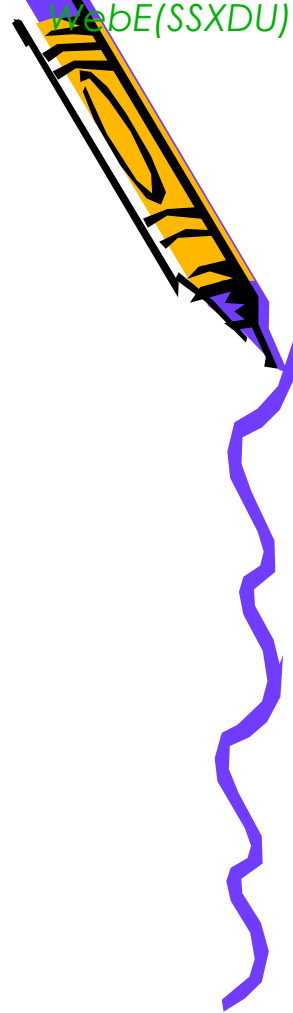
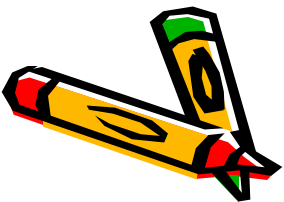
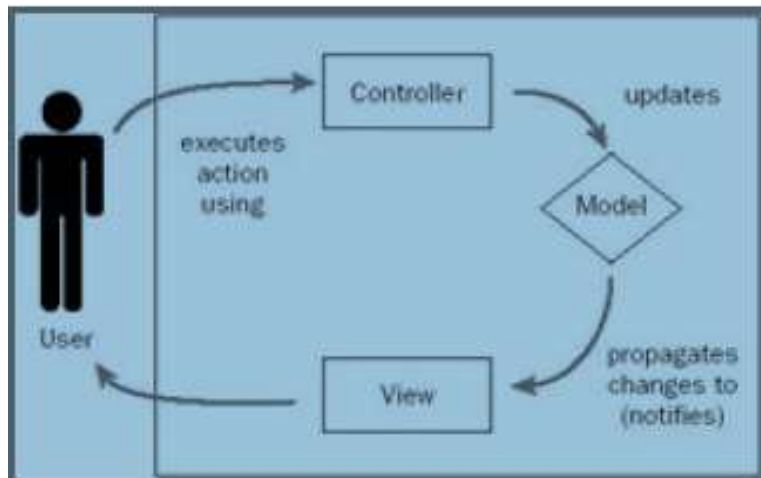
Spring MVC开发示例--前端页面

```
<%@ taglib prefix="c" uri=http://java.sun.com/jstl/core %>
<html>
    <head>
        <title>Hello world!</title>
    </head>
    <body>
        <h1>Hello world</h1>
        <p>
            Right now, the time is <c:out value="${date}"/>.
        </p>
    </body>
</html>
```

- 此处的welcome.jsp对应了ModelAndView的构造参数，<c:out value="\${date}"/>使用了C标签，取出了后台controller传来的名为data的属性，并显示出来。



Spring MVC开发示例--整体回顾



Spring和持久化框架的整合

- 和Web框架不同，Spring没有自己开发的持久化开发（这体现了“不重复发明轮子”的Spring思想）。
- Hibernate：时下最流行的O/R映射框架，也是第一个Spring整合的第一个O/R映射框架。
- JDO：Spring支持JDO 1.1和JDO 2.0标准。JDO的几个厂商也有自己的JDO Spring集成。
- TopLink：TopLink是市场上最古老的O/R映射框架。如今已是Oracle旗下的产品，并且Oracle已经让它和Hibernate、JDO一样好用。
- Apache OJB：Apache的O/R映射框架
- iBATIS：严格来说，iBATIS不是一个O/R产品。但是它可以同时SQL语句实现对象到参数的映射和result set到对象的映射。



Spring和Hibernate的整合

- Hibernate是一个开源的O/R（关系型数据库和对象）映射框架。对JDBC进行了轻量级的封装。
- Hibernate可以应用在任何使用JDBC的场合，既可以在Java的客户端程序使用，也可以在Servlet/JSP的Web应用中使用。
- Hibernate使用的是对象查询语句，而不是传统的SQL语句，但和SQL语句相似，很好理解。Hibernate也支持原始的SQL语句，因为HQL不足以支持所有的SQL功能。
- 通过在配置文件application.xml中配置Hibernate所需的bean就可以在Spring中使用Hibernate了。
- Hibernate封装数据库一般分为三层，第一层是实体层，第二层是DAO层，第三层是service层



Spring Hibernate开发示例--application.xml

- 配置数据源：

```
<bean id="dataSource"  
class="org.springframework.jdbc.datasource.DriverManagerDataSource">  
    <property name="driverClassName"  
value="dataSource.driverClassName" />  
    <property name="url" value="dataSource.url" />  
    <property name="username" value="dataSource.username" />  
    <property name="password" value="dataSource.password" />  
</bean>
```

- dataSource的配置属性一般放置在properties文件中
(方便配置)



Spring Hibernate开发示例--application.xml

- 配置sessionFactory：这里使用注解声明型的sessionFactory，即实体通过注解声明（也可以使用xml描述文件定义实体，但很麻烦）。

```
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
    <property name="dataSource"><ref bean="dataSource" /></property>
    <property name="hibernateProperties">
        <props><prop
key="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</prop></props>
    </property>
    <property name="packagesToScan">
        <list>
            <value>实体所在package</value>
        </list>
    </property>
</bean>
```

- sessionFactory需要使用dataSource，并配置相关属性，hibernate.dialect表示hibernate方言，与具体的数据库对应，packagesToScan表示实体类所在的包。

Spring Hibernate开发示例--application.xml

- 配置transactionManager: 为hibernate配置事务管理

```
<bean id="transactionManager"  
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
```

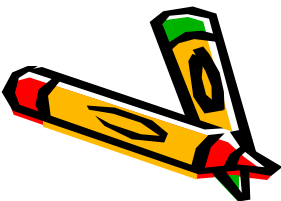
```
    <property name="sessionFactory" ref="sessionFactory"  
/>
```

```
    <property name="dataSource" ref="dataSource" />
```

```
</bean>
```

```
<tx:annotation-driven transaction-manager="transactionManager"  
proxy-target-class="true" />
```

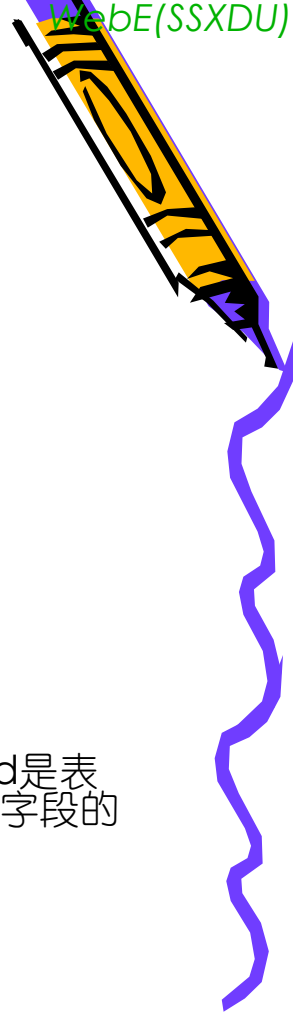
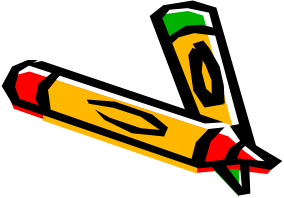
- transactionManager需要使用dataSource和transactionManager
- tx:annotation-driven声明了事务管理会在代码中以注解的方式实现。



Spring Hibernate开发示例——实体

```
@Entity
@Table(name="users")
public class User implements Serializable
{
    @Id
    private int id;
    @Column(unique=true,name="username")
    private String userName;
    @Column(name="password")
    private String userPassword;
}
```

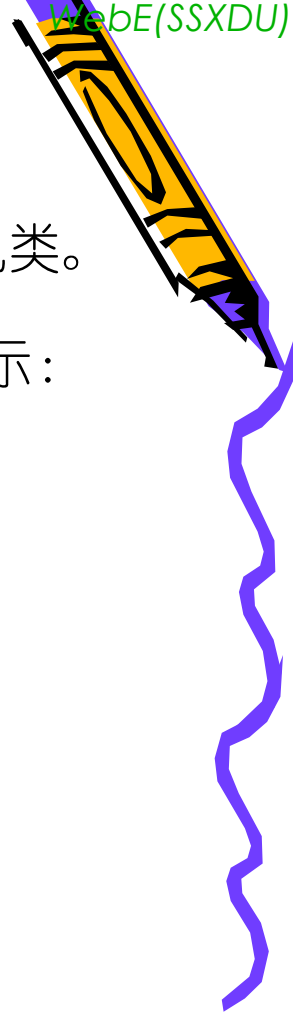
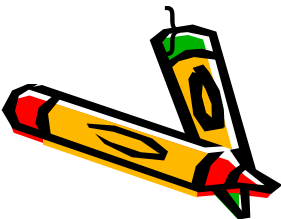
@Entity注解表示这是一个实体，@Table注解了该实体对应的数据库表，@Id表明id是表的PK，@Column表示普通字段，name属性对应了表中的字段名，unique表示改字段的唯一的



Spring Hibernate开发示例--DAO

- DAO (data access object) 一般分为两层，接口和实现类。仅仅是接口暴露给service层
- DAO接口和普通的接口一样，不需要特定的注解。如下所示：

```
public interface UserDao {  
    public List<User> getAll();  
    public User getByld(Integer id);  
    public void add(User user);  
    public void delete(Integer id);  
    public void edit(User user);  
    public User getByUsername(String username);  
}
```



Spring Hibernate开发示例--DAO

- DAO层接口的实现由以下UserDaoImpl类完成

@Repository("usersDaoImpl")

```
public class UserDaoImpl implements UserDao{
```

```
    @Resource(name= "sessionFactory")
```

```
    private SessionFactory sessionFactory;
```

```
    public List<Users> getAll() {
```

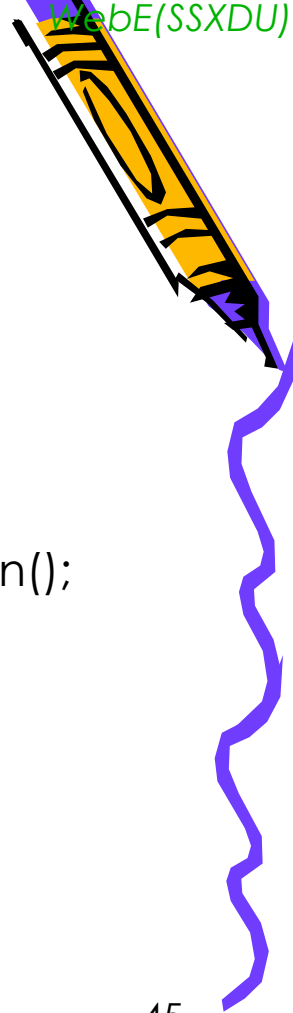
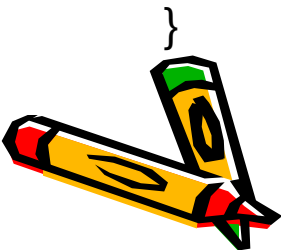
```
        Session session = sessionFactory.getCurrentSession();
```

```
        Query query=session.createQuery("From Users");
```

```
        return query.list();
```

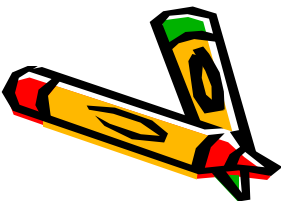
```
    }
```

```
//忽略其他方法的实现
```



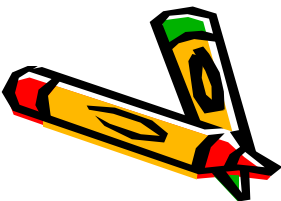
Spring Hibernate开发示例--DAO

- 在DAO实现中，最前面需要声明
`@Repository("usersDaoImpl")`，表示这是一个提供数据操作的bean，`SessionFactory`需要用`@Resource(name="SessionFactory")`注解，这样Spring会自动寻找与`SessionFactory`名字匹配的bean，为其提供依赖注入。
- 在`UserDaoImpl`中有一个`@Resource(name="SessionFactory")`，这是注解就是根据名字“`SessionFactory`”寻找bean，为变量`SessionFactory`提供依赖注入。
- Hibernate访问数据库，要通过Session，即由`SessionFactory`创建，Session再创建查询Query，Query可以执行HQL语句；Session也可以创建SQLQuery，可以执行SQL语句，前面说过，HQL还无法完成SQL的所有功能。



Spring Hibernate开发示例--service

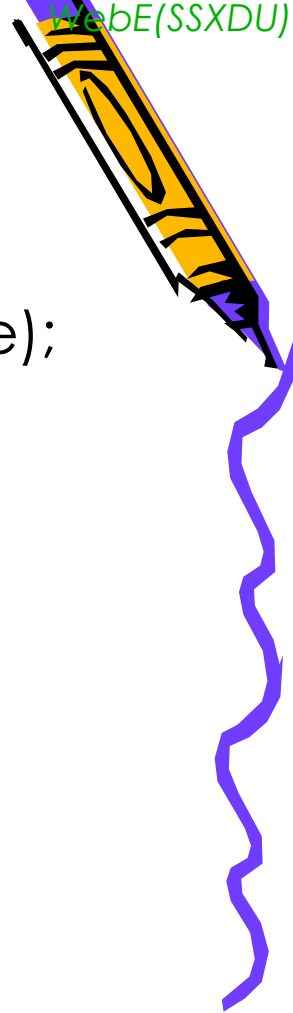
- Service层封装了DAO。它的主要功能将controller提供的数据处理成DAO可以处理的数据。
- Service直接向应用程序提供服务，应避免将DAO直接暴露给用户。
- service层一般也分成两部分，一部分是api接口，另一部分是接口的实现，向外只暴露接口（体现了面向接口编程的思想）。



Spring Hibernate开发示例--service

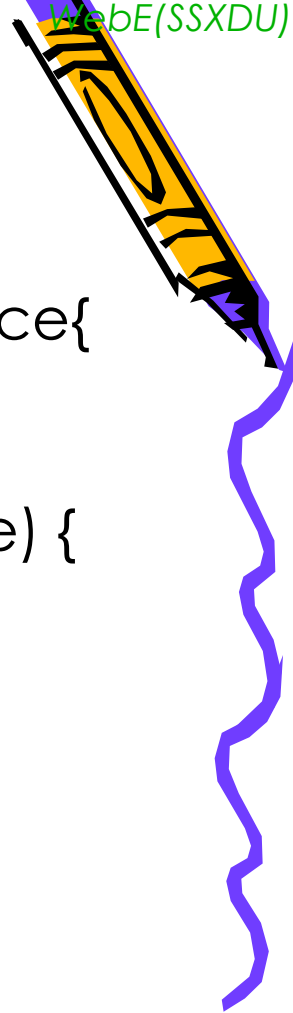
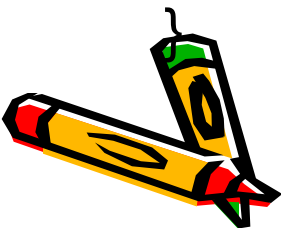
```
public interface UserService {  
    public Users getByUsername(String username);  
    public void addUser(Users user);  
    public Users getById(int id);  
    public void updateUser(Users user);  
    public void delete(Users user);  
    public List<Users> getAll();  
}
```

service接口和普通的接口一样，不需要特殊的注解。



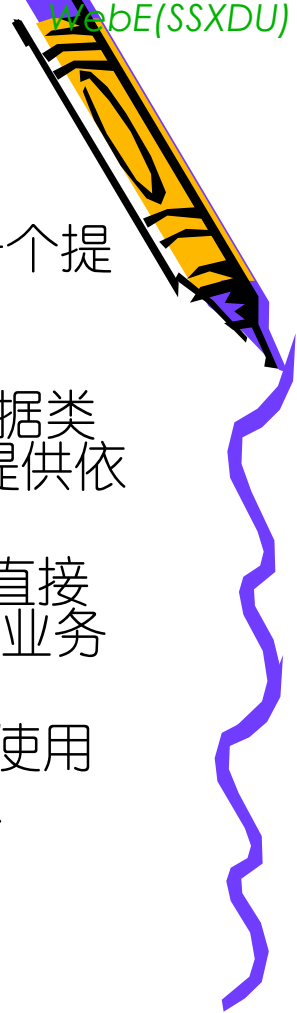
Spring Hibernate开发示例--service

```
@Service("userService")
public class UserServiceImpl implements UserService{
    @Autowired
    private UserDao userDao;
    public Users getByUsername(String username) {
        return
        userDao.getByUsername(username);
    }
    //忽略其他方法的实现
```



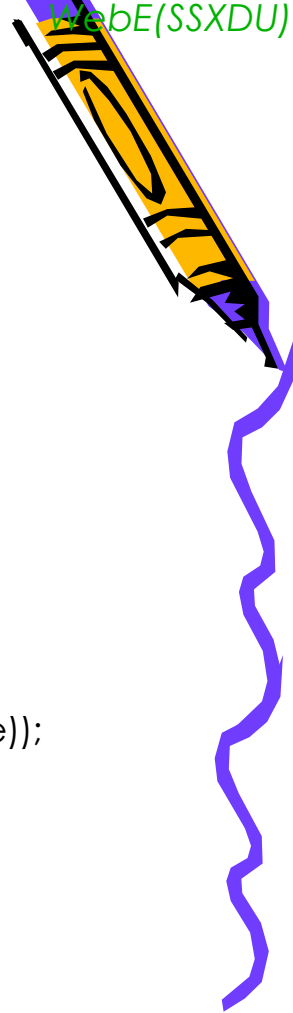
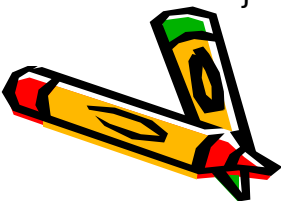
Spring Hibernate开发示例--service

- @Service("userService")注解表示UsersServiceImpl是一个提供服务服务的bean。在controller中，就可以根据“userService” 依赖注入该bean。
- 在属性usersDao 前注解@Autowired，表示该变量可以根据类型自动依赖注入。它会自动寻找到UserDaoImpl，为变量提供依赖注入。
- 在service的方法实现中，例如getByUsername，service直接调用了DAO的方法。在一般的service中，并没有很复杂的业务逻辑代码。
- 至此，Hibernate的数据封装就完成了。Controller就可以使用service来完成数据操作了。在之前的SpringMVC中，加入hibernate数据操作。



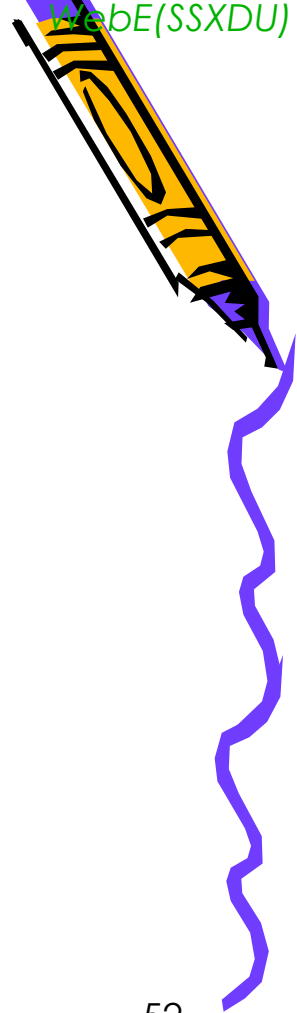
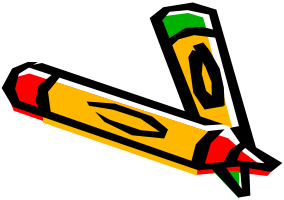
Spring Hibernate开发示例--controller

```
@Component
public SimpleController
{
    @Resource(name="userService")
    UserService userService;
    @RequestMapping (value="/welcome",method={RequestMethod.GET})
    public ModelAndView handleRequestInternal(HttpServletRequest request,HttpServletResponse response)
    throws Exception
    {
        String username=request.getParamiter("username");
        ModelAndView mav = new ModelAndView("welcome");
        mav.addObject("user", userService. getByUsername(username));
        return mav;
    }
}
```



Spring Projects

SPRING BOOT



Spring Boot

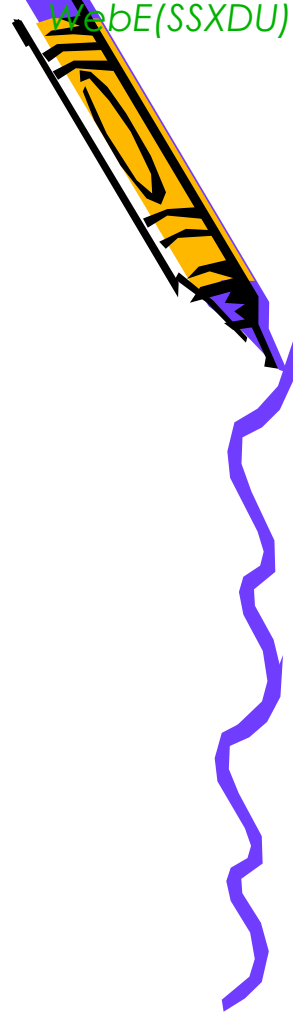
- Spring Boot 旨在创建独立的、产品级的基于Spring的应用程序。
。 Spring boot的最大特点是可以快速构建一个Spring应用程序。
。 Spring Boot的特征如下：
- 1.创建独立的Spring应用程序；
- 2.内嵌了Tomcat或Jetty，因此，应用程序不需要打包成war。
- 3.提供了"starter" POMS依赖声明，这是Boot特有的，可以大大简化Maven中pom.xml的配置。
- 4.自动完成Spring的配置，无需复杂的xml文件配置，如果需要修改配置，则在项目的src/main/resource中添加application.properties（Spring Boot Reference Guide里有相应属性的含义）



Spring Boot——创建Maven项目

- 1、创建maven-archetype-quickstart项目
- 2、设置项目坐标
- 3、在pom.xml文件里添加以下依赖

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.0.2.RELEASE</version>
</parent>
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
</dependencies>
```



Spring Boot——主程序

- 因为spring boot是普通的java项目，其内嵌有服务器，所以需要有一个启动程序。启动Spring Boot项目和运行普通Java程序类似。

编写主程序如下：

```
@Controller
```

```
@EnableAutoConfiguration
```

```
public class SampleController {
```

```
    @RequestMapping("/")
```

```
    @ResponseBody
```

```
    String home() {
```

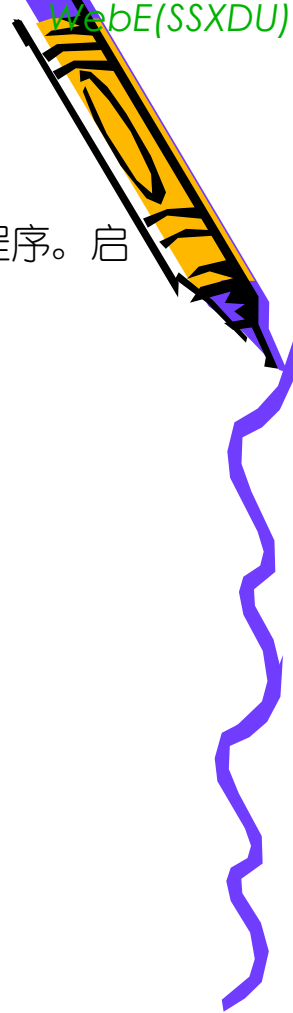
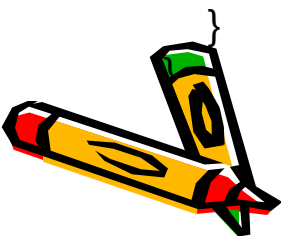
```
        return "Hello World!";
```

```
}
```

```
public static void main(String[] args) throws Exception {
```

```
    SpringApplication.run(SampleController.class, args);
```

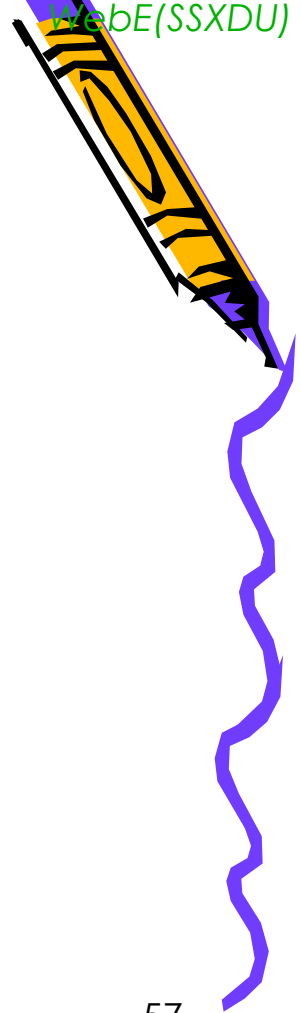
```
}
```



Spring Boot——主程序

- 在Spring Boot中，需要用@EnableAutoConfiguration注解一个含有main函数的类，此类可以单独编写，在前面的例子中，它被放在了一个controller中。
- main函数中的SpringApplication.run(xxx.class, args)，会识别@EnableAutoConfiguration，自动配置应用程序（如果有application.properties文件，则使用其中的配置）。
- application.properties包含有spring项目的配置信息，spring boot项目启动时，会自动读取其中的配置信息，详细内容参见Spring Boot Reference Guide附录。
- 搭建完架构之后的业务逻辑处理代码和传统Spring项目一样。





END

