**Tecket 1**

**1.Фазы jsf: Invoke Application и Render Response（jsf 阶段：调用应用程序和渲染响应）**

JSF(JavaServer Faces)它是一个基于服务器端组件的用户界面框架。它用于开发Web应用程序

JSF(JavaServer Faces)应用程序的生命周期从客户端对页面发出HTTP请求时开始，并在服务器响应页面时结束。

Jsf is java server faces，and it is a standard framework which can build java web application.

It start when it sends http request and ends when the server responds to the page

JSF生命周期分为两个主要阶段：

* 执行阶段
* 渲染阶段

**调用应用阶段**

在此阶段，JSF处理应用程序级事件，例如提交表单或链接到另一个页面(处理通过单击按钮、链接等触发的事件)

之后，JSF实现将控制转移到“渲染响应”阶段。

First stage invoke application，it does application-level events，for example submit a form or link to another page（something like push the bottom or link），after these it will turn into the render response

**渲染响应阶段**

这是JSF生命周期的最后阶段。 在此阶段，JSF 使用在前面的步骤中接收到的数据生成响应,使用managed bean更新页面视图，之后生成html页面并发送给用户(错误收集在message中)

**2.Способы задания конфигурации в Spring （spring配置)**

将 Spring 配置到应用开发中有以下三种方式：

1.基于xml配置：applicationContext.xml配置文件。所有的bean、对象创建或者类型都是我们自己手动输入,且必须在创建ClassPathXmlApplicationContext时指定xml文件

2.基于java配置：您需要创建一个带有注解Configuration的类，并在其中通过添加带有各种注解的方法（例如bean）来配置 spring。

3.基于注解配置:spring可以只用注解配置.

**Tecket 2**

**1.MVC в JSF**

M:JSF 中的Model 是一个Managed bean，包含业务逻辑并传递给用户的 Java 类。 为它们指定了注释，并借助这些注释配置了它们的行为。

V:JSF 中的View是用 xhtml 扩展名编写的布局。这是一种非常方便的格式，因为浏览器可以理解它，并且还可以通过添加库来扩展它。

C:我们不用在JSF中手写Controller，因为它是由框架实现的，我们只需要在布局中指定动作，框架就会明白我们想要得到什么。

**2. Главные аннотации в CDI beans java EE(CDI beans java EE 的主要注释)**

CDI Beans 有很多注解：

@NoneScoped - 未定义上下文，其他 beans 管理生命周期；

@RequestScoped - 上下文 - 请求；

@ViewScoped - 页面上下文（该组件在访问页面时创建一次，并且只要用户在页面上就被使用）；

@SessionScoped - 上下文 - 会话；

@ApplicationScoped - 上下文 - 应用程序；

@Named - 用来给bean起个名字，然后就可以在jsf页面上使用了。 通常，此注释与范围注释一起使用

（随便记几个就好了）

**Tecket 3**

1. **Spring Web MVC: View Resolvers (Spring Web MVC：查看解析器)**

SPRING 中的 MVC 实现：

servlet调度程序接收请求，然后查看其设置以确定将请求发送到哪个controller。接下来，在controller中处理请求。处理后，响应到达调度程序。

根据接收到的数据，调度器寻找他需要的视图（ViewResolver）。

模型数据被传递给view（或者如果需要的话传递给model），然后view被发送给用户。

View Resolver 是由能够通过名称查找视图的对象实现的接口。本地化支持也是可能的。

**2.[Angular: ключевые особенности, отличия от AngularJS](https://docs.google.com/document/d/1lNUjNyu-ybeYOqlt1ypfulWUxTpSIYkuaBl4qFK_Ipc/edit" \l "heading=h.u0nn9s9oqqqq)(Angular：主要特性，与 AngularJS 的区别)**

Angular和AngularJS一样，实现了MVVM模型

但AngularJS 目前已被弃用，因为它存在性能问题、文档混乱以及大量冗余工具。

AngularJS有一个严格的组件框架，它们很难重用，Angular有一个组件层次结构，这使得重用它们变得容易和简单。AngularJS 是通用的，但安全性和可管理性要差得多。

AngularJS使用 JavaScript，Angular使用TypeScript（这意味着更少的代码、更高的安全性，并且通常是一种更现代的语言）。

Angular也适用于弱移动设备（AngularJS 中没有）。

**Tecket 4**

**1.[Профили и платформы Java EE(Java EE 配置文件和平台)](https://docs.google.com/document/d/1ppajsRY07bSFRMGKpDzI1Bf8cSWsfX4i8rgthi81DoI/edit" \l "heading=h.waahpvfl3n5h)**

JavaEE 中有两个配置文件：

Web Profile - 仅包含Web应用程序工作所需的那些组件，这些组件是 Servlets、JSP、JSF、JPA、CDI、EJB。

Full Profile是它的Java的完整集合，它包括了Web Profile,除了Web，它还有JAX-RS、JAX-WS、JAXB、JNDI等等。

JavaEE平台：

JME-（Java Micro Edition）是一个 API 和一个要求最低的 VM，用于在智能手机/平板电脑上开发和启动应用程序。 基于 JSE 的早期版本，因此某些功能（例如泛型）不起作用

JSE-（Java 标准版）关注于提供 Java 本身的基本标准特性，它绝对定义了一切：语言的基本类型和对象、高级类、应用程序网络性能和安全性。 包括包：java.util、java.math、java.io、java.nio、java.lang 等。

JEE-（Java Enterprise Edition）用于开发企业应用程序。 它建立在 JSE 平台之上，还支持开发更大、更复杂、分层和更安全的程序。 包含：WebSocket、JSF、统一 EL、RESTful Web 服务 API、DI、EJB、JPA 和 Java 事务 API。

**2.[Типы DI в Spring](https://docs.google.com/document/d/1ppajsRY07bSFRMGKpDzI1Bf8cSWsfX4i8rgthi81DoI/edit" \l "heading=h.xfc4cqw9jyj)(Spring中的DI类型)**

Spring实现了三种类型的依赖注入：

1.Constructor-based DI -容器将使用bean参数调用构造函数，然后将其注入到类中

2.基于Setter的DI -首先，容器将调用不带参数的bean构造函数，然后它将调用标有@Autowired注释的setter并将必要的依赖项推送到那里（@Autowired Car(Wheel w)...）

3.Field-based DI -容器会通过反射将依赖push到class的fields中(不推荐使用，反射很吃资源，很容易违反Single Responsibility原则，很难debug)

**Tecket 5**

1. **Spring MVC: обработка запросов, DispatcherServlet**

**(Spring MVC：请求处理和DispatcherServlet)**

Spring MVC 的整个逻辑是围绕DispatcherServlet构建的，它接收并处理所有HTTP请求和对它们的响应.收到请求后，会发生以下事件链：

1、DispatcherServlet访问HandlerMapping接口，该接口决定调用哪个controller

2. 控制器接收请求并调用一个实用方法，该方法将视图的名称返回给调度程序

3.使用ViewResolver，调度器决定使用哪个View

4. View创建后，dispatcher将数据以属性的形式发送给model，最终已经显示在浏览器中

**2.Single Page Application(SPA): преимущества, недостатки(单页应用程序（SPA）：优点，缺点)**

这是一个 Web 应用程序，它使用单个 HTML 文档作为所有网页的包装器，并动态加载 HTML、CSS、JS 代码，操作交互、数据交互通常通过 AJAX

好处：

1. 由于大量现成的库和框架，易于创建。

2. 简单的数据缓存，因为它们是在一个请求中加载的。

3. 工作速度，因为主要部分资源已经加载完毕，页面只加载必要的数据，节省了时间。

缺点：

1. 没有JS，无法使用应用的全部功能

2. 浏览器无法索引页面 (SEO)，因为 服务器不会立即生成/呈现内容，而是在已加载的默认页面上动态生成/呈现

**Tecket 6**

**1. Технология RMI. Использование RMI в Java EE(RMI技术.在 Java EE 中使用 RMI)**

Java远程方法调用(Java Remote Method Invocation)系统允许运行在一个 Java 虚拟机中的对象调用运行在另一个Java 虚拟机中的对象的方法, 通过TCP工作。一般情况下，对象都是按值(拷贝)传递的，所以传递的对象必须是Serializable（可系列化的）。

要使 RMI 工作，您需要：

服务器端：

1.创建界面

2.实现它并创建一个对象

3.获得 stub（存根对象，将交给客户端，通过RMI协议实现通信）

4.创建注册表（客户端将从中接收对象的注册表）

5.将创建的存根绑定到注册表（绑定按名称进行）

客户端：

1.创建与服务器相同的接口（如果此接口对客户端和服务器都可用，则不创建）

2.连接到服务器注册表

3.获得正确的存根

4.附加到接口

5.调用需要的方法

在 Java EE 中，RMI 用于EJB客户端和bean之间的远程通信，即远程接口EJB是一个RMI存根。

**2.Управление состоянием в React. Flux & Redux(React中的状态管理。Flux & Redux)**

Flux 是一种用于构建React应用程序的架构，它描述了如何存储、修改和显示全局数据（状态）。

基本概念：

Dispatcher从view(例如，从界面按钮)接收事件，并将它们发送到数据存储(Store)进行处理。

Store是一个存储数据的单例。只有Store知道如何更改数据。您不能直接从React组件更改它们。数据改变后，Store 发送事件给视图，视图被重绘。

Redux 是一个实现了简化的 Flux 模式的小型库。 Redux 中没有 Dispatcher，但是有一个store—一个单例，整个应用的状态的store。使用纯函数(reducer)进行状态更改

**Tecket 7**

**1.[JNDI. JNDI в Java EE. Способы взаимодействия с JNDI. Их преимущества и недостатки.](https://docs.google.com/document/d/1ppajsRY07bSFRMGKpDzI1Bf8cSWsfX4i8rgthi81DoI/edit" \l "heading=h.l2mqvwc35j96)(JNDI。 Java EE 中的 JNDI。 与 JNDI 交互的方法。 他们的优点和缺点。)**

JNDI 是一种通过名称访问对象的API。组织为命名和目录服务。

大多数情况下，JNDI 用于企业开发。主要用例是设置对数据库的访问。应用程序只知道JNDI 名称，例如“java:comp/env/Db”，连接细节本身由管理员在 Web 容器中描述。

JNDI 支持命名和目录服务的不同实现。 其中一些是：DNS、RMI、LDAP、COBRA。

JNDI 的好处：

- 易于应用程序设置

- 数据库密码与应用程序分开，更安全

- 更改数据库时（例如，从测试到生产），您不需要重建应用程序

缺点：

- 容器依赖

- 使用旧版本的 log4j 时存在基于 jndi 的漏洞

**2.React. Особенности. Архитектура(React特点和结构)**

React 是一个用于开发用户界面的 JS 库。 允许您使用道具（属性）和状态（数据）创建自己的组件。 组件以 HTML 呈现。

特点： - 从父组件到子组件的数据传输

- 虚拟 DOM

- JSX 是 JS 里面有 HTML

React 负责表示界面，不强加任何特定的架构。在编写处理大量数据的复杂应用程序时，通常会使用Flux架构和Redux库。

**Tecket 8**

**1. Платформы Java. Сходства и различия. (Java 平台.共同点和不同点)**

有 3 个最著名的 API：JME、JSE、JEE

JME-（Java Micro Edition）是一个 API 和一个要求最低的VM，用于在智能手机/平板电脑上开发和启动应用程序。基于 JSE 的早期版本，因此某些功能（例如泛型）不起作用

JSE-（Java 标准版）关注于提供 Java 本身的基本标准特性，它绝对定义了一切：语言的基本类型和对象、高级类、应用程序网络性能和安全性。 包括包：java.util、java.math、java.io、java.nio、java.lang等。

JEE-（Java Enterprise Edition）用于开发企业应用程序。 它建立在 JSE 平台之上，还支持开发更大、更复杂、分层和更安全的程序。 包含：WebSocket、JSF、统一 EL、RESTful Web 服务 API、DI、EJB、JPA和Java事务 API。

所有 Java 平台都支持Java语言的全部功能，区别仅在于是否存在某些API。

### **2.Двухфазные и трехфазные конструкторы в Spring и Java EE(Spring和Java EE 中的两阶段和三阶段构造函数)**

Spring中的两阶段构造函数：常规构造函数+带有@PostConstruct注解的方法。常规构造函数将首先被调用，然后是标记的方法。注解是 Java EE 的一部分，且在Spring中工作。

Spring中的一个三阶段构造函数（第三阶段，使用前改变对象）：首先有一个原生的构造函数，然后处理依赖，调用处理依赖的对象构造，然后添加slices。

slices允许您以与配置bean时数据更改相同的方式更改行为。这样的构造函数有自己的作用域。它应该用于创建自定义bean。

**Tecket 9**

**1. [Валидаторы в JSF. Создание, назначение и тд.](https://docs.google.com/document/d/1ppajsRY07bSFRMGKpDzI1Bf8cSWsfX4i8rgthi81DoI/edit" \l "heading=h.mzz6m4mdgfvw)(JSF中的验证器。创建、任命等)**

JSF 中的验证器是验证器接口的实现。validate方法采用FacesContext、UiComponent和要验证的值(Object)。

可以使用<f:validator/>标签将验证器附加到组件

为了让JSF注册自定义验证器，使用了@FacesValidator注释。有3个最著名的API：JME、JSE、JEE

JME-（Java Micro Edition）是一个 API 和一个要求最低的 VM，用于在智能手机/平板电脑上开发和启动应用程序。 基于 JSE 的早期版本，因此某些功能（例如泛型）不起作用

JSE-（Java 标准版）关注于提供 Java 本身的基本标准特性，它绝对定义了一切：语言的基本类型和对象、高级类、应用程序网络性能和安全性。包括包：java.util、java.math、java.io、java.nio、java.lang等。

JEE-（Java Enterprise Edition）用于开发企业应用程序。它建立在JSE平台之上，还支持开发更大、更复杂、分层和更安全的程序。 包含：WebSocket、JSF、Unified EL、API for RESTful web services、DI、EJB、JPA 和 Java Transaction API。

还有内置验证器：BeanValidator（调用 bean 上的方法进行验证）、DoubleRangeValidator、LengthValidator、RegexValidator、RequiredValidator。

**2.Реализация контроллера в Spring Web MVC(在Spring Web MVC中实现控制器)**

Controller是一个用@Controller（或RestController）注解的特殊类。Controller的任务是拦截传入的请求，以所需格式打包数据，将此数据发送到所需模型，然后将模型的响应传递回 DispatcherServlet。控制器方法上挂了各种注解，例如@GetMapping、@PostMapping（这是针对请求路径的）和用于从请求中接收数据的注解，例如@PathVariable 或@ResourcesVarible。

**Tecket 10**

**1.JSF Restore View phase(JSF视图形成阶段)**

视图形成阶段（RestoreView），在这个阶段，框架（JSF）做了以下事情：

如果这是客户端的第一个请求，则创建对象，分配监听器、验证器、转换器，所有这些都放在FacesContext中并发送以创建对用户的响应。

如果FacesContext已经存在（这不是第一个请求，并且所有对象都已经创建），则检查请求类型，然后是渲染新页面（如果是 GET）和创建/更改数据（如果是POST）开始。

### 2.**Spring framework. Отличия от Java EE(**Spring framework。与Java EE的区别)

粗略地说，Java EE是一个构造函数，它是模块化的，您可以制作自己的程序集，连接和断开非常小的模块。它有许多视图、bean等的实现。使其成为开发单一的、可扩展的应用程序的理想选择。

Spring也分模块，但是这些模块都比较大，相辅相成比分开住更方便。 Spring 对如何编写应用程序施加了更严格的限制，在某些情况下这非常重要。此框架适用于小型 Web 应用程序或微服务架构。同样的ajax，在JavaEE中用1行编写（感谢 JSF），在spring中你需要手动编写它。

**Tecket 11**

**1. [Java EE CDI Beans прерывание жизненного цикла (Interception)(](https://docs.google.com/document/d/1ppajsRY07bSFRMGKpDzI1Bf8cSWsfX4i8rgthi81DoI/edit" \l "heading=h.15ax757ftmr)Java EE CDI Beans 生命周期中断（拦截）)**

CDI 规范提供了一种“挂钩”机制，允许您在应用程序代码中实现面向方面的编程技术。 换句话说，开发人员可以使用端到端功能，即在引用 CDI bean 的某些方法之前，程序员有机会执行预处理、业务方法本身和后处理。为了这些目的，有一个 CDI 拦截器机制。

如何在被调用的方法前后注入额外的逻辑：

创建自定义注解，上面需要指定@InterceptorBinding

接下来，创建一个将充当拦截器的类，将我们创建的注释 + @Interceptor 挂在上面

创建一个方法进行处理，带InvocationContext ctx参数，用@AroundInvoke注解标注。 在方法本身中，为了调用方法，我们编写了ctx.proceed();。在之前或之后，我们可以描述额外的逻辑。

现在剩下的就是将我们自定义的注解添加到我们要扩展其逻辑的方法上（如果你在类上挂注解，那么它的所有方法都会被中断）

**2.Компоненты React. State and props. "Умные" и "Глупые" компоненты(react组件。state和props。“聪明”和“愚蠢”的组件)**

React 实现了组件重用的方法。我们可以创建组件，然后在页面（按钮、导航栏、页脚、表单等）上重用它们

您可以通过两种方式创建组件：创建一个扩展 React.Component 的类，或创建一个返回 View的函数。智能组件存储特定状态并可以根据它进行更改。愚蠢的人不存储任何东西，只是在屏幕上显示一些东西。智能组件能够发出请求、处理信息等。

愚蠢的人专注于 UI，并且主要使用 props 来实现可重用性。

每个组件都有自己的状态和道具。 它们需要跟踪某些状态的变化或灵活性（将信息从组件传递到组件）

**Tecket 12**

**1 JavaServer Faces. Особенности, недостатки, преимущества.(JSF：主要特性、优点、缺点)**

JSF 是一个Java Web应用程序框架，它是 Java EE 标准的一部分。

特点：

- 面向组件的结构。 该界面由可以相互嵌套的组件构建而成。呈现为 HTML 元素。

- 模板 (Facelets) 是用 XHTML 编写的。使用自定义元素、属性。

- 业务逻辑在Java bean中呈现

- 内置AJAX支持。

- 写在Servlet API之上

好处：

- 将业务逻辑与表示分离

- 程序员需要编写更少的JS代码

- 易于实施AJAX

劣势

- 不能很好地扩展

- 不适用于高性能应用

- 学习曲线（learning curve）

**2.CDI бины - что это, зачем нужны, если есть EJB и ManagedBeans(CDI-beans：当有 EJB 和Managed Bean 时，它们是什么以及为什么需要它们)**

CDI bean 是允许开发人员使用依赖注入概念的特殊bean。CDI使管理bean成为可能。与 MB 不同，CDI bean更加强大和灵活，它们可以使用拦截器、原型、装饰器等。另一方面，EJB具有一些CDI所不具备的特性（例如，事务函数、计时器、异步）。但是，总的来说，EJB和CDI是相似的，甚至可以相互注入。

**Tecket 13**

**1.Построение интерфейсов на JSF. Иерархия компонентов JSF.(在JSF中构建页面。JSF组件的层次结构。)**

JSF网络应用程序界面在XHTML文件中进行了描述。可以有常规的HTML元素（div、p、h1、img）和JSF组件。组件是派生自UIComponent的类。它们形成了一个层次结构。每个组件（UIViewRoot 除外）都有父组件，也可以有子组件。组件的优点是它们可以将复杂的布局和逻辑封装在单个XHTML标记后面的JS中。有第三方组件库，例如PrimeFaces和IceFaces，可以使用丰富的现成组件轻松构建界面。

**2.Java EE CDI Beans: принципы инъекции бинов.(Java EE CDI Beans：bean注入原理)**

CDI是依赖注入的标准。为了让它工作，您首先需要一个管理bean的CDI容器。Beans可以依赖于其他beans（例如，某种StudentsResource可以依赖于Database）。使用@Inject注释，我们可以要求CDI容器找到合适的bean并将其传递给我们。依赖注入与类字段（甚至私有字段，通过反射）和构造函数（依赖项在构造函数中传递）一起使用。适当的 bean 选择机制会考虑请求的类或接口、bean名称(@Named)和备选方案(@Alternative)。

值得注意的是，注入到bean中的不是原始依赖bean类，而是容器本身动态创建并允许您实现拦截器的代理类。

**Tecket 14**

**1.Managed bean: назначение, конфигурация, использование в xhtml(Managed bean：用途、配置、xhtml 中的用法)**

Managed Beans有一个所谓的“生命区”（作用域）——这是bean将被创建并且可用于处理请求和执行其任务的时间。 范围是：

RequestScoped -将在每个HTTP请求中创建bean实例。

SessionScoped-bean将在会话期间可用，即实例将在客户端连接时创建一次

ViewScoped -在访问页面时创建一次

应用程序范围-在服务器启动时创建一次并一直存在到服务器关闭

CustomScoped -我们可以自己管理这个范围。 为此，必须在需要时将bean放置在地图中

NoneScoped - bean未绑定到任何范围。可应用于辅助垃圾箱。它的生命周期由第三方bean 控制。

您可以通过注释（@ManagedBean、@SessionScoped）或通过 faces-config xml 文件配置 bean（包括它们的范围）：所需bean的名称、类、范围和托管属性都填在那里。但是，当然，通过注释更容易。

在 JSF 中，您可以通过 EL:#{myBean.property} 访问 ManagedBean

你不应该在 JSP 中使用它们，你不应该混合使用两个不同的框架。

现在 ManagedBean 被认为已过时，它们被 CDI bean 取代。

**2.Архитектура и состав Spring Web MVC(Spring MVC的架构和组成)**

Spring MVC是一个用于编写基于Spring核心的Java EE Web应用程序的框架。

Spring MVC的核心是DispatcherServlet。它是一个接受所有请求并将控制传递给程序员编写的控制器的 servlet。

HandlerMapping -用于查找正确控制器的接口

Controller是一个带有@Controller注解的处理请求的类。 它实现了一些用于数据准备的业务逻辑。

ViewResolver - 用于寻找合适视图的接口

查看（view）——生成一个HTML页面。 大多数情况下，这是一个用JSP、Thymeleaf或 Freemaker编写的模板。控制器生成模型并将其传递给视图。在模板中，我们可以读取模型属性并将其显示在页面上。

**Tecket 15**

**1. Контекст управляемых бинов. Конфигурация контекста бина（Managed bean 上下文。 Bean上下文配置）**

上下文决定了bean将绑定到什么：整个应用程序、会话、请求、JSF页面。您可以通过 ApplicationScoped、SessionScoped、RequestScoped、ViewScoped注释或通过faces-config.xml 进行配置：<managed-bean-scope>application</managed-bean-scope>

另外对于Managed beans：

ManagedBeans可以拥有ManagedProperty（Managed属性），框架可以向其中注入某些初始值（字符串、数字、枚举）。这是通过ManagedProperty注释或在 faces-config.xml 文件中配置的

bean本身由JSF框架管理。它有自己的FacesContext上下文，（如果你真的想要）你可以通过编程方式提取所需的bean

**2.Шаблоны MVVM и MVP. Сходства и отличия от MVC（MVVM 和 MVP 模式。 与MVC的异同）**

MVP，即 Model-View-Presenter，是比 MVC 晚得多创建的模板。随着web编程工具的发展，已经没有必要将controller单独归类，而是出现了Presenter。它包括负责渲染和更新 View 的部分，以及负责更新 Model 的部分。简单来说就是包含了接口逻辑，负责Model和View的同步。

MVVM—模型-视图-视图模型—是另一种模式。 重点是ViewModel与View没有直接关系，而是使用简单的命令与之通信。这种模式推荐用于实现数据绑定（DataBinding）的项目—视图和模型之间的直接连接，无需使用控制器或展示器。ViewModel本身包含一个转换为视图的模型，以及视图访问模型的命令。

**Tecket 16**

**1.[REST в спринге: методы и аргументы(](https://docs.google.com/document/d/1ppajsRY07bSFRMGKpDzI1Bf8cSWsfX4i8rgthi81DoI/edit" \l "heading=h.vj3jcy1mk1sd)Spring 中的 REST：方法和参数)**

REST 可以使用 @Controller 注释在常规 Spring MVC 控制器中实现。

对于在正文中返回序列化响应的每个处理程序方法，您需要添加 @ResponseBody 注释。

为方便起见，Spring 引入了@RestController 注解，它默认将@ResponseBody 应用于所有标有@RequestMapping 注解的方法。

请求处理程序方法可以通过@PathVariable 注释接受来自 URL 的参数，通过@RequestParam 获取请求参数（查询字符串），通过@RequestBody 从请求主体反序列化数据。 Spring会根据Content-Type头自动判断数据格式，如果有合适的库就反序列化（Jackson用的是JSON）

**2.[Навигация в React. React Router](https://docs.google.com/document/d/1ppajsRY07bSFRMGKpDzI1Bf8cSWsfX4i8rgthi81DoI/edit" \l "heading=h.j79ce85lia23)（在React中导航。React Router）**

React Router是一个路由系统，允许您在组件之间导航，还允许您将应用程序请求映射到特定组件。

三个主要对象：

Router定义一组路由，当请求到达应用程序时，Router将请求与路由匹配。如果某个路由与请求URL匹配，则选择该路由来处理请求。

并且还定义了一个Routes对象用于路由选择。它包含一组路由，并允许您选择第一个可用路由并将其用于处理。

每条路线代表一个路线对象。它有许多属性。特别地，这里为路由设置了两个属性：

路径：请求的URL将与之匹配的地址模式

element -负责处理此路由请求的组件

**Tecket 17**

**1.[Конвертеры JSF, создание и назначение(](https://docs.google.com/document/d/1ppajsRY07bSFRMGKpDzI1Bf8cSWsfX4i8rgthi81DoI/edit" \l "heading=h.lkpqm3iujb7c)JSF 转换器、创建和赋值)**

JSF 中的转换器是实现转换器接口的类。 他们的任务是将对象转换为字符串，反之亦然。 方法：getAsObject和getAsString。要注册您的转换器，您需要在faces-config.xml中注册它或使用FacesConverter注释

在JSF中的用法：

<f:converter>

有内置转换器：IntegerConverter、DoubleConverter、DateTimeConverter、EnumConverter 等。

附加信息：

JSF提供了一组转换器来将值从组件转换为给定的格式。实现 javax.faces.convert.Converter 接口。有基本数据类型的标准转换器（DoubleConverter、LongConverter等）。使用转换器属性或使用子标签自动分配转换器（基于数据类型）。

此外，我们可以通过实现 Converter 接口来创建我们自己的类转换器。

如果出现错误，您可以指定要显示的消息。

数据转换器在组件中用于将数据转换为给定格式。对于主要的数据类型，已经有标准的转换器可以连接到一个特殊的标签（例如，<convertDateTime />。你可以通过在你的类中实现 Converter接口来创建你自己的转换器。你需要实现 getAsObject ()（将输入转换为对象）和 getAsString方法，然后在faces-config.xml中注册转换器或使用@FacesConverter(name)注解对其进行注解。

**2.[Реализация model в Spring web MVC](https://docs.google.com/document/d/1ppajsRY07bSFRMGKpDzI1Bf8cSWsfX4i8rgthi81DoI/edit" \l "heading=h.iz8wl2ohlh0)(Spring web MVC中的实现模型)**

Spring MVC中的模型是一个描述具有模型属性的映射的接口。模型的主要方法有addAttribute、getAttribute、asMap。您可以将任何对象放入模型中，并通过字符串键获取它。 通过模型，视图访问需要在页面上显示的应用程序数据。

有@ModelAttribute 注释，它允许您在控制器中设置一个方法，用属性填充模型，然后所有处理程序都可以使用这些属性。您还可以将处理程序参数标记为@ModelAttribute。在这种情况下，我们的对象将使用请求中的参数创建，然后放入模型中。

**Tecket 18**

**1.[CDI Beans](https://docs.google.com/document/d/1ppajsRY07bSFRMGKpDzI1Bf8cSWsfX4i8rgthi81DoI/edit" \l "heading=h.gmykfs4t1miy)**

CDI bean - 满足以下要求的类：该类必须是静态的（并且不是嵌套的非静态的）并且必须具有构造函数（不带参数或带参数，但标有@Inject 注释）

CDI bean 从 Java EE 6 开始就存在，并且支持比JSF ManagedBeans更多的特性CDI bean 和它们的对应物一样，支持依赖注入bean的生命周期由CDI容器管理，也就是说，如果需要，它们的实例会自动创建。

**2.[Angular DI](https://docs.google.com/document/d/1ppajsRY07bSFRMGKpDzI1Bf8cSWsfX4i8rgthi81DoI/edit" \l "heading=h.omed8m9kd1rk)**

Angular框架的一个显着特点是支持依赖注入。你可以编写自己的类，用@Injectable 注解标记并将它们注入到其他类、服务、组件中。大多数情况下，实现的服务实现了与表示无关的业务逻辑。例如日志记录、与 API 的通信。使用 DI 的好处是，如果需要，可以很容易地用另一个实现替换一个特定的实现，测试一个，从而单独测试系统组件。

**Tecket 19**

**1.[Шаблоны и представление в Angular](https://docs.google.com/document/d/1ppajsRY07bSFRMGKpDzI1Bf8cSWsfX4i8rgthi81DoI/edit" \l "heading=h.3wq8nlqp9w3y) （Angular 中的模板和视图）**

模板用于定义组件的显示方式Angular 模板就像修改过的 HTML支持插值：{{value}}

属性语法扩展：[attr]=”value”——单向绑定

[(attr)]=”value”——双向绑定

@event=”handler”——事件处理器

对于条件渲染，使用 \*ngIf 指令，用于循环 - \*ngFor。

Angular 中的视图是一种树状数据结构，包含对 DOM 元素的引用。

**2.[Dependency Lookup Spring](https://docs.google.com/document/d/1ppajsRY07bSFRMGKpDzI1Bf8cSWsfX4i8rgthi81DoI/edit" \l "heading=h.cdkctjt3uxr4)（依赖查找Spring）**

Spring 中有两种类型的依赖注入：Dependency Injection和Dependency Lookup在 DI 中，一切都很简单 - spring 自己做所有事情，我们只需用 @Autowired 注释标记您需要提供所需 bean的“位置”而Spring自己也在寻找容器，各取所需但是，有时候，DI可能会失败（例如，获取错误的bean）那么我们就可以自己去寻找必要的依赖了。

例子：

ApplicationContext appContext = newClassPathXmlApplicationContext (“/application-context.xml”);

MyBean bean = appContext.getBean("myBean");

也就是说，我们自己获取 xml 并从那里自己加载必要的bean但他们几乎从不这样做。

**Tecket 20**

**1.[Process validations phase, Update module values phase（流程验证阶段，更新模块值阶段](https://docs.google.com/document/d/1ppajsRY07bSFRMGKpDzI1Bf8cSWsfX4i8rgthi81DoI/edit" \l "heading=h.oi0yw9ehhw6e)）**

执行检查，在这个阶段所有的字符串值都被转换为对象那些还没有通过校验的对象在这里通过为此，调用 UIViewRoot#processValidators() 方法。

在更新模型值阶段，您可以确保所有对象都经过验证和检查，之后“树”中的所有对象都被传输到模型，从而更新服务器上的数据为此（很奇怪），调用了 UIVIewRoot#processUpdates() 方法。

**2.[Жизненный цикл Spring-приложения](https://docs.google.com/document/d/1ppajsRY07bSFRMGKpDzI1Bf8cSWsfX4i8rgthi81DoI/edit" \l "heading=h.jrpyycqozk8h)（Spring应用程序生命周期）**

1.创建一个容器

2.容器读取配置

3. beans的描述从配置和类路径加载（带注解）

4.Bean实例化

5.正在进行依赖注入

6.调用PostConstructor

7.回调和BeanPostProcessor被调用

8.应用启动

9.应用关闭

10.PreDestroy，调用回调

11.申请停止

**Tecket 21**

**1.Реализация Ajax в JSF（在 JSF 中实现 Ajax）**

在 JSP 中实现 Ajax 请求有两种方法：旧的和新的。

旧：使用 jsf.ajax.request() JavaScript API提供标准查询桥和粒度控制示例：<h:commandButton… onClick=”jsf.ajax.request(this, event, {execute:'myinput', render:'outtext''}); return false；”/>

新：<f:ajax> 标签非常方便，无需接触 JavaScript示例：<h:commandButton>

<f:ajax execute=”@form” render=”outtext”/>

</h:commandButton>

在这两种情况下，渲染属性负责重新渲染页面的组件之一以反映更改。

**2.CDI beans: контекст (Bean Scope) （ CDI bean：上下文（Bean 作用域））**

@NoneScoped - 未定义上下文，其他 beans 管理生命周期；

@RequestScoped - 上下文 - 请求；

@ViewScoped - 页面上下文（该组件在访问页面时创建一次，并且只要用户在页面上就被使用）；

@SessionScoped - 上下文 - 会话；

@ApplicationScoped - 上下文 - 应用程序；

@CustomScoped - 组件被创建并存储在 Map 类型的集合中生活区由程序员管理。

bean 的上下文决定 bean 的生命周期是否绑定到整个应用程序、用户会话、请求或从属 bean可以通过注解配置作用域：ApplicationScoped、SessionScoped、RequestScoped、ConversationScoped、Dependent（默认使用）

**Tecket 22**

**1.Spring MVC: handler mapping（Spring MVC：处理程序映射）**

HandlerMapping是Spring MVC的核心组件之一，用来保存request-handler之间的映射。  
简单来说，request指的是请求地址（还包括请求方法等），handler指的是Controller中对应的方法。

**2.JSX. Применение в реакте. Пример синтаксиса（JSX。在react中的应用。语法示例）**

 JSX，是一个 JavaScript 的语法扩展。我们建议在 React 中配合使用 JSX，JSX 可以很好地描述 UI 应该呈现出它应有交互的本质形式。JSX 可能会使人联想到模板语言，但它具有 JavaScript 的全部功能。

JSX 可以生成 React “元素”，React 使用 JSX 来替代常规的 JavaScript。

我们知道元素是构成 React 应用的最小单位，JSX 就是用来声明 React 当中的元素。

与浏览器的 DOM 元素不同，React 当中的元素事实上是普通的对象，React DOM 可以确保 浏览器 DOM 的数据内容与 React 元素保持一致。

要将 React 元素渲染到根 DOM 节点中，我们通过把它们都传递给 ReactDOM.render() 的方法来将其渲染到页面上：

## React 实例

var myDivElement = <div className="foo" />;

ReactDOM.render(myDivElement, document.getElementById('example'));

**Tecket 23**

**1.Фаза Apply request values JSF（应用请求值 JSF 阶段）**

**2.Реализация IoC и CDI в Spring（Spring中的IoC和CDI实现）**

IOC是“Inversion of Control”的缩写，翻译过来就是“****控制反转****”。

Spring中的体现就是：把创建对象的过程交给Spring来进行管理，从而做到将原来需要自己手动new对象，变成直接从Spring中获取。

这就就好比Spring中有一个容器，我们将Bean放到这个容器中，让这个容器为我们创建实例，当需要时我们直接从这个容器中进行获取即可。这个容器的实现理念就是IOC。

Spring实现IOC容器的是通过：工厂+反射，实现的。

首先，您需要创建一个beans.xml文件，您可以在其中为所有创建的bean设置。然后，使用 Inject 注释，将依赖项注入构造函数。

**Tecket 23**

**1. Компоненты в Angular: взаимодействие с представлениями и сервисами（Angular 中的组件：与视图和服务交互）**

**2.Инициализация Spring Beans（初始化Spring Bean）**

在Spring中，Bean的初始化有两种方式：

1、在配置文档中指定init-method属性来完成。

2、实现org.[springframework](https://so.csdn.net/so/search?q=springframework&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/u010142437/article/details/_blank).beans.factory.InitializingBean接口。