vmware | SPRING & CLOUD FOUNDRY
开发者大会

中国北京 - 2012年12月7日-8日

Spring 多客户端开发

Josh Long

VMWare 旗下 SpringSource 部门的 ***Spring 开发人员技术布道师***

http://www.joshlong.com || @starbuxman || josh.long@springsource.com

关于 springsource

- **SpringSource** 是开发出业界领先的企业 Java 框架 **Spring 框架**的组织。
- **VMware** 在 2009 年收购了 **SpringSource**
- **VMware** 和 **SpringSource 带您开始云中旅行，并以**“构建、运行、管理”为使命
  - 已与 **Adobe**、**SalesForce** 和 **Google** 等行业巨头**建立了合作伙伴关系，帮助大家跨多平台为** Spring 用户提供最佳体验
- 是 **Apache HTTPD** 和 **Apache Tomcat 等项目的主要贡献者**

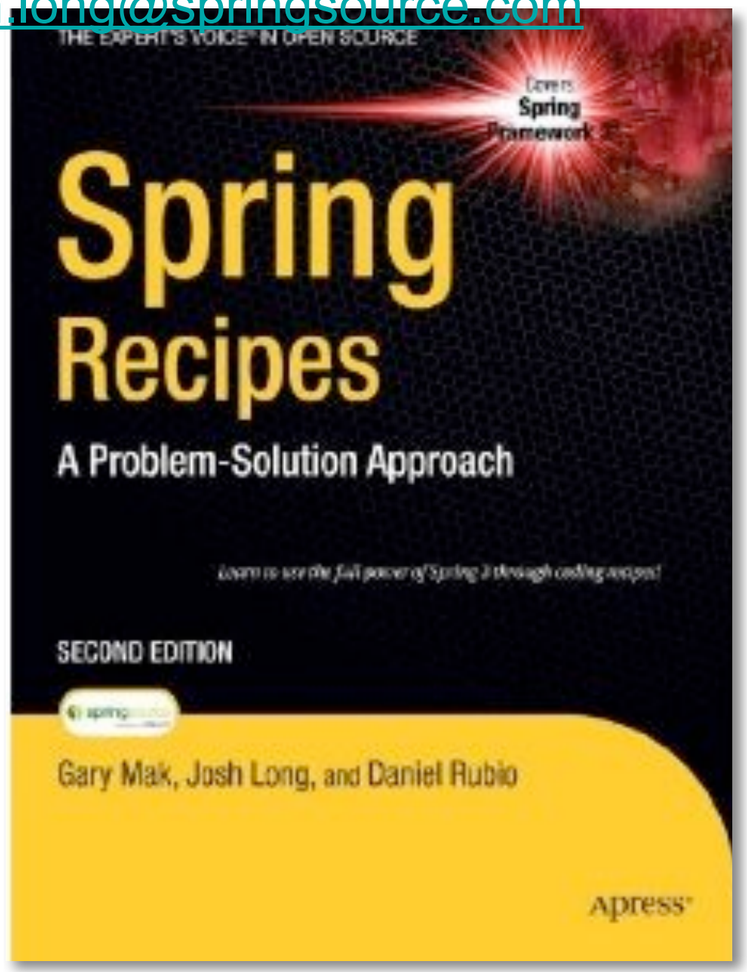
Josh Long (龙之春) 介绍

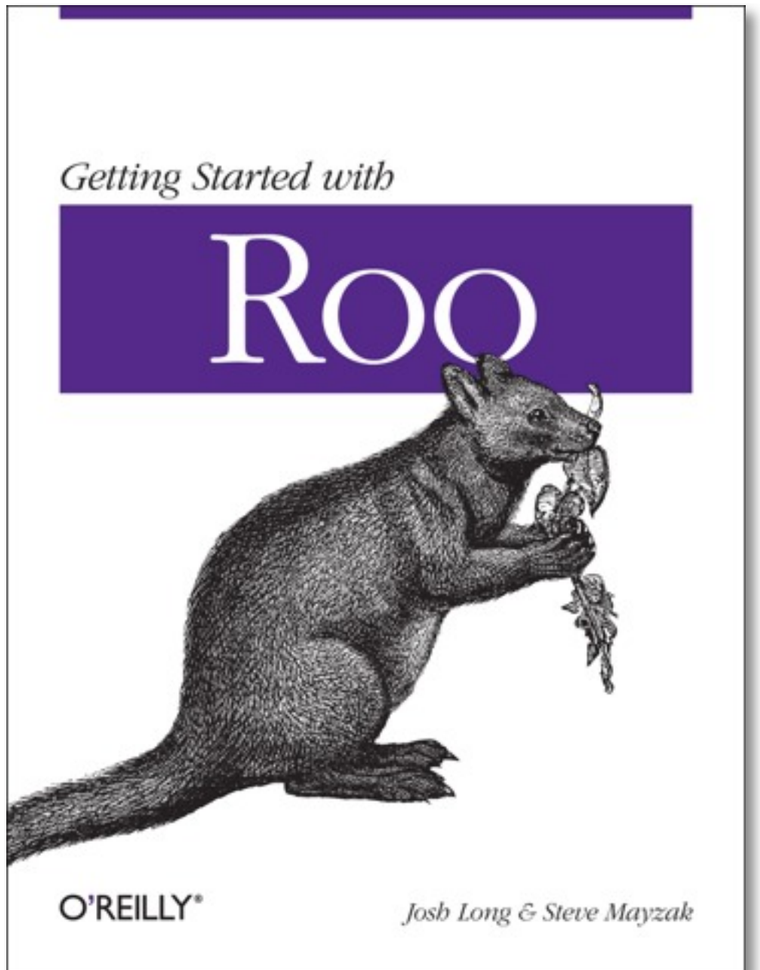
Spring 开发人员技术布道师

twitter：@starbuxman

weibo：@springsource

josh.long@springsource.com





Josh Long 介绍

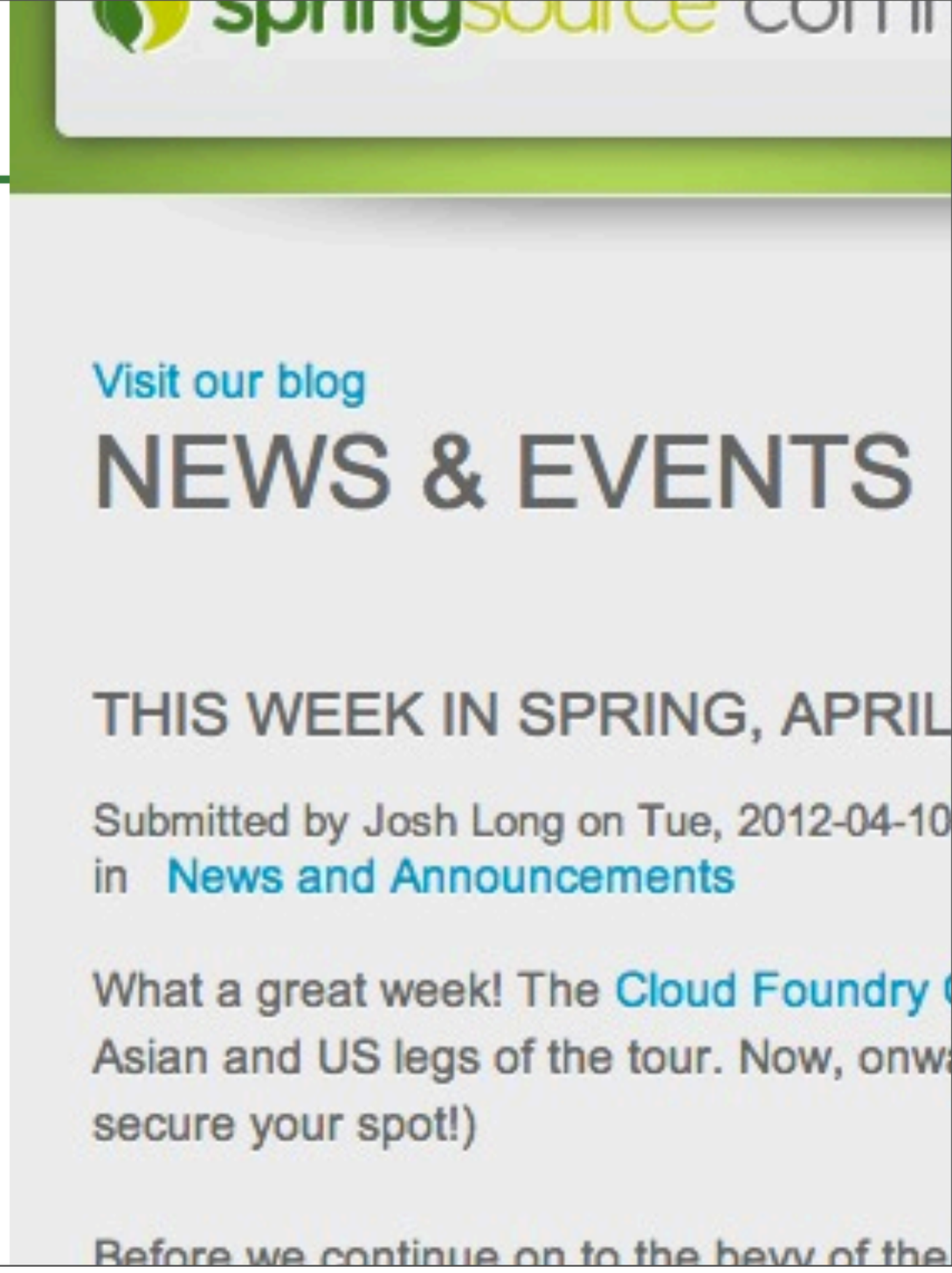
Spring 开发人员技术布道师

twitter：@starbuxman

josh.long@springsource.com

参与贡献的项目：

- Spring Integration
- Spring Batch
- Spring Hadoop
- Activiti 工作流引擎
- Akka Actor 引擎



我们为什么在这里？

> “软件实体（类、模块、功能等）应该”

-Bob Martin

我们为什么在这里？

千万不要
彻底改造车轮！

Spring 的目标：

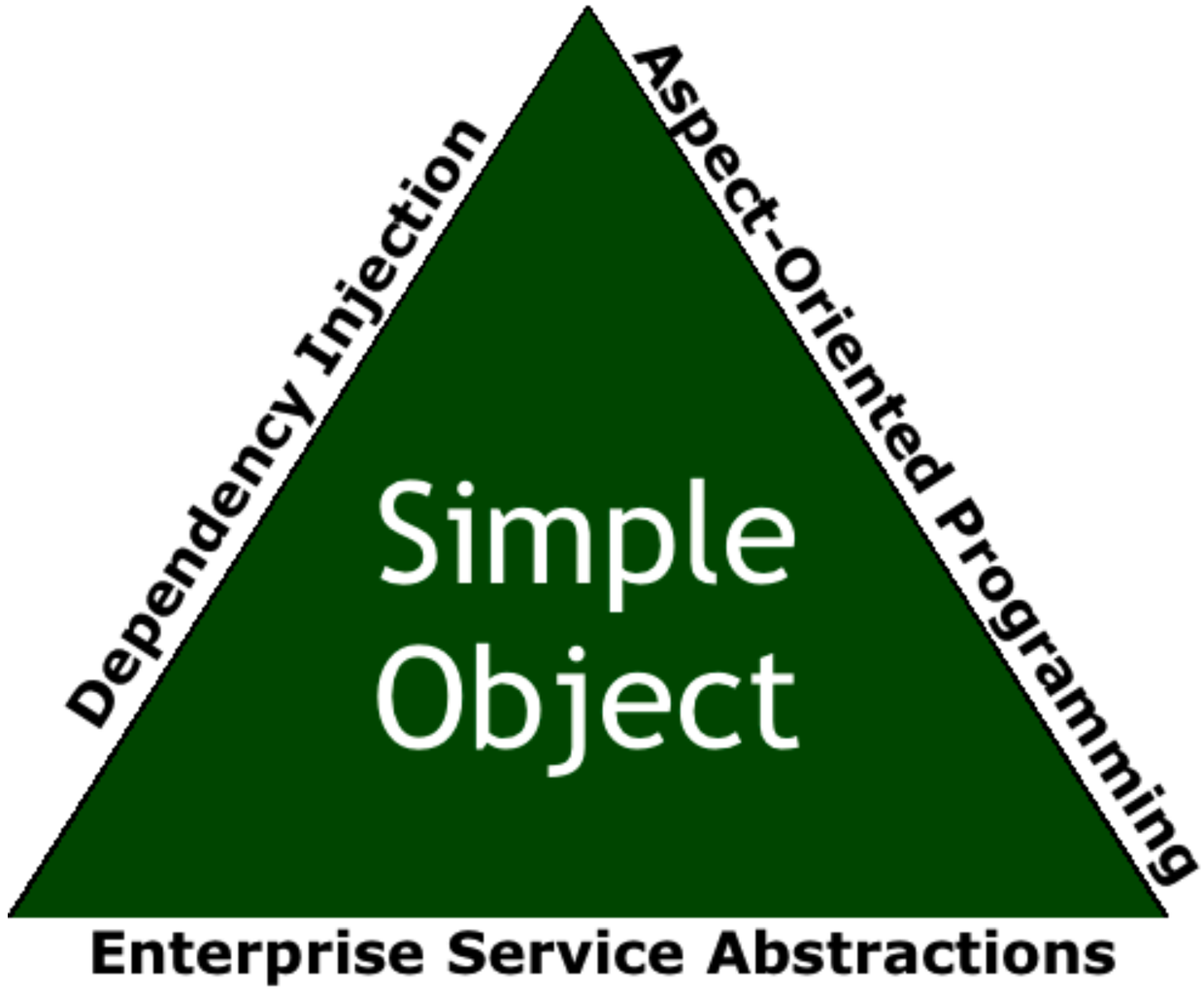
将简单性引入 java 开发

Web 层与 RIA	服务层	批处理	集成与消息传递	数据访问 /NoSQL/ 大数据	移动

Spring 框架

云：	轻量	传统
CloudFoundry Google App Engine Amazon BeanStalk 其他	tc Server Tomcat Jetty	WebSphere JBoss AS WebLogic （在旧版本上也是如此！）

并非所有方面都平衡


Dependency Injection
Aspect-Oriented Programming
Simple Object
Enterprise Service Abstractions

Spring 的多客户端开发

Cloud Foundry Primer 上的 Spring

使用 Cloud Foundry Spring 命名空间（生成者）

<cloud:redis id="redis"/>

<cloud:data-source id="dataSource"/>

```
<bean class="org.sf.orm.jpa.LocalContainerEntityManagerFactoryBean"
id="entityManagerFactory">
      <property name="dataSource" ref="dataSource"/>
</bean>
```

访问绑定到 Cloud Foundry 的服务（生成者）

```
@Configuration(“cloud”)
public class MyServicesConfiguration {

    private String mongoDatabaseServiceName = "survey-mongo";

    @Bean
    public CloudEnvironment cloudEnvironment() {
      return new CloudEnvironment();
    }

    @Bean
    public MongoServiceInfo mongoServiceInfo() {
      return cloudEnvironment().getServiceInfo(
          mongoDatabaseServiceName, MongoServiceInfo.class);
    }

    @Bean
    public MongoDbFactory mongoDbFactory() {
      MongoServiceCreator msc = new MongoServiceCreator();
      return msc.createService(mongoServiceInfo());
    }
}
```

访问绑定到 Cloud Foundry 的服务（使用）

@Inject private **DataSource dataSource** ;

@Inject private **MongoTemplate mongoTemplate;**

演示

- 借助 Cloud Foundry 将应用程序放到云端

Spring Web 支持

3.1 新增功能：无 XML 的 Web 应用程序

- 在 servlet 3 容器中，您也可以使用 Java 配置，而无需 web.xml：

```
public class SampleWebApplicationInitializer implements WebApplicationInitializer {

   public void onStartup(ServletContext sc) throws ServletException {

      AnnotationConfigWebApplicationContext ac =
            new AnnotationConfigWebApplicationContext();
      ac.setServletContext(sc);
      ac.scan( “a.package.full.of.services”, “a.package.full.of.controllers” );

      sc.addServlet("spring", new DispatcherServlet(webContext));

   }
}
```

Thin、Thick、Web、移动和 Rich 客户端：Web 核心

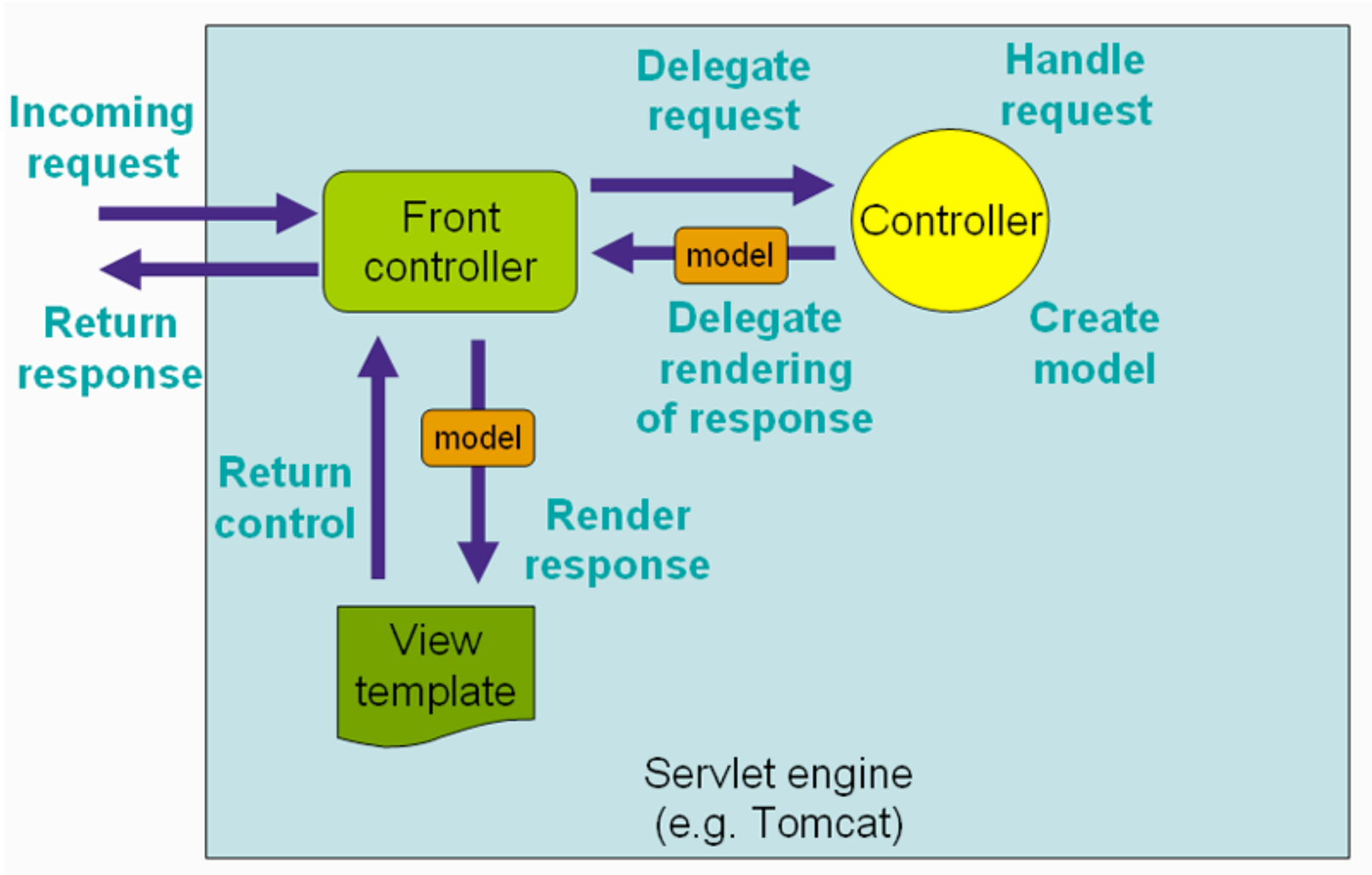
Spring Dispatcher Servlet

- 对象不一定要特定于 Web。
- Spring Web 支持较低级别的 Web 机制：
  - **HttpRequestHandler**（支持远程：Caucho、Resin、JAX RPC）
  - **DelegatingFilterProxy**。
  - **HandlerInterceptor** 打包发送至 **HttpRequestHandlers** **的请求**
  - **ServletWrappingController** 使您能够强制发送至 servlet 的请求从 Spring Handler 链中通过
  - **OncePerRequestFilter** 可以确保无论应用多少次筛选，每个操作仅发生一次。提供了避免重复筛选的良好途径
- Spring 使用 **WebApplicationContextUtils** 提供对 Spring 应用程序上下文的访问；**WebApplicationContextUtils** 具有可查询上下文的静态方法，即使是在 Spring 未管理 Web 组件的环境中也是如此

Thin、Thick、Web、移动和 Rich 客户端：Web 核心

- **Spring 可提供用于集成您选择的 Web 框架的最简单方法**
  - Spring 面向 JSF 1 和 2
  - 针对 Struts 1 的 Struts 支持
  - Tapestry、Struts 2、Stripes、Wicket、Vaadin、Play 框架等
  - GWT、Flex

Thin、Thick、Web、移动和 Rich 客户端：Spring MVC



Spring MVC 控制器的构造

Spring MVC 配置 – 配置

```
@Controller
public class CustomerController {


}
```

Spring MVC 控制器的构造

Spring MVC 配置 – 配置

```
@Controller
public class CustomerController {

    @RequestMapping(value=”/url/of/my/resource”)
    public String processTheRequest() {
        // ...
        return “home”;
    }

}
```

GET http://127.0.0.1:8080/**url/of/my/resource**

Spring MVC 控制器的构造

Spring MVC 配置 – 配置

```
@Controller
public class CustomerController {

    @RequestMapping(value=”/url/of/my/resource”,
                    method = RequestMethod.GET)
    public String processTheRequest() {
        // ...
        return “home”;
    }

}
```

GET http://127.0.0.1:8080/**url/of/my/resource**

Spring MVC 控制器的构造

Spring MVC 配置 – 配置

```
@Controller
public class CustomerController {

    @RequestMapping(value="/url/of/my/resource",
                    method = RequestMethod.GET)
    public String processTheRequest( HttpServletRequest request) {
        String contextPath = request.getContextPath();
        // ...
        return "home";
    }

}
```

GET http://127.0.0.1:8080/**url/of/my/resource**

Spring MVC 控制器的构造

Spring MVC 配置 – 配置

```
@Controller
public class CustomerController {

    @RequestMapping(value=”/url/of/my/resource”,
                    method = RequestMethod.GET)
    public String processTheRequest( @RequestParam(“search”) String searchQuery ) {
        // ...
        return “home”;
    }

}
```

GET http://127.0.0.1:8080/**url/of/my/resource?search=Spring**

Spring MVC 控制器的构造

Spring MVC 配置 – 配置

```
@Controller
public class CustomerController {

    @RequestMapping(value=”/url/of/my/{id}”,
                    method = RequestMethod.GET)
    public String processTheRequest( @PathVariable(“id”) Long id) {
        // ...
        return “home”;
    }

}
```

GET http://127.0.0.1:8080/**url/of/my/2322**

Spring MVC 控制器的构造

Spring MVC 配置 – 配置

```
@Controller
public class CustomerController {

    @RequestMapping(value="/url/of/my/{id}" )
    public String processTheRequest( @PathVariable("id") Long customerId,
                                     Model model ) {
        model.addAttribute("customer", service.getCustomerById( customerId ) );
        return "home";
    }

    @Autowired CustomerService service;

}
```

GET http://127.0.0.1:8080/**url/of/my/232**

Spring MVC 控制器的构造

Spring MVC 配置 – 配置

```
@Controller
public class CustomerController {

    @RequestMapping(value=”/url/of/my/resource”,
                    method = RequestMethod.GET)
    public String processTheRequest( HttpServletRequest request, Model model) {
        return “home”;
    }

}
```

Spring MVC 控制器的构造

Spring MVC 配置 – 配置

```
@Controller
public class CustomerController {

    @RequestMapping(value=”/url/of/my/resource”,
                    method = RequestMethod.GET)
    public View processTheRequest( HttpServletRequest request, Model model) {
        return new XsltView(...);
    }

}
```

Spring MVC 控制器的构造

Spring MVC 配置 – 配置

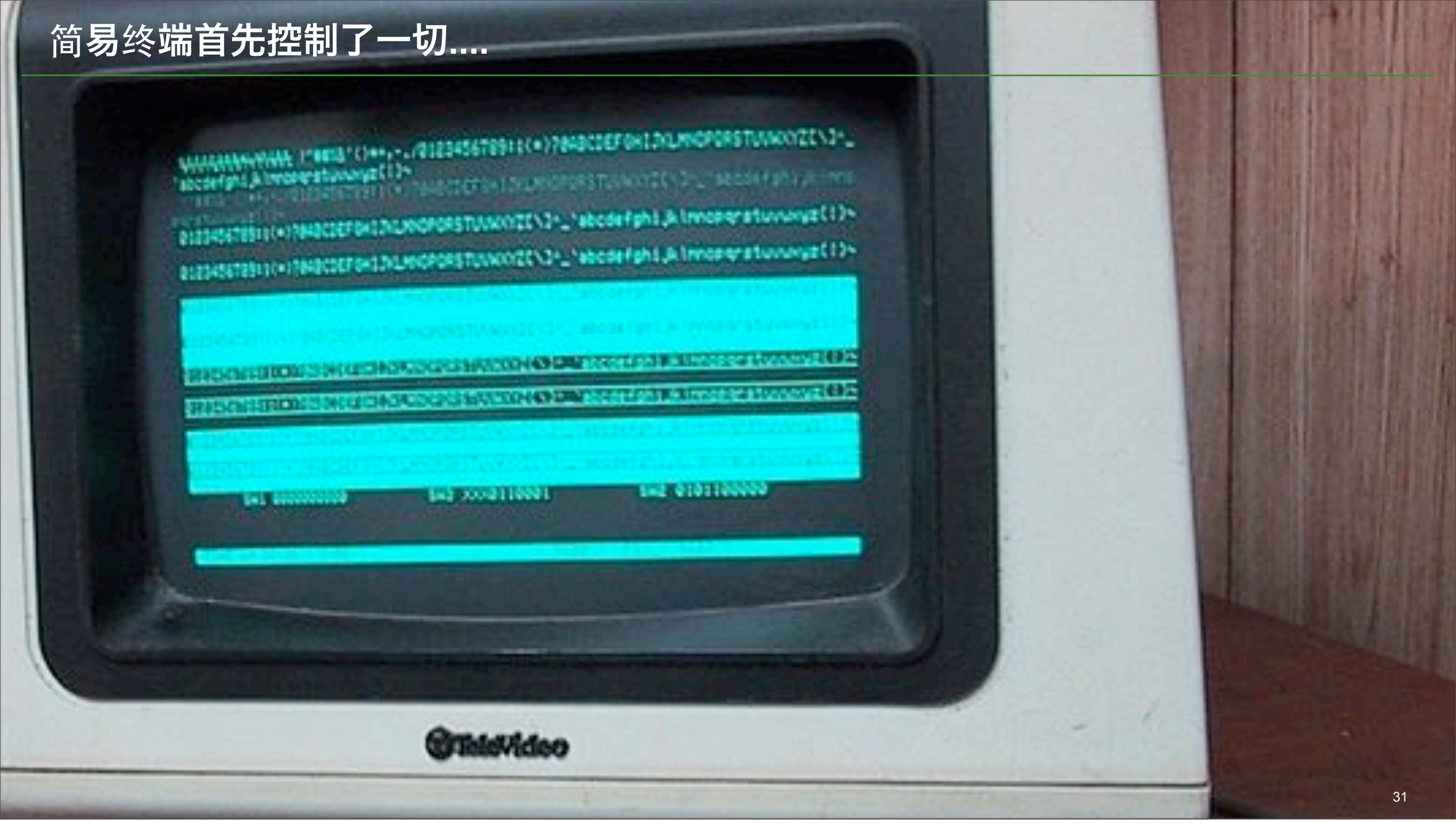
```
@Controller
public class CustomerController {

    @RequestMapping(value=”/url/of/my/resource”,
                    method = RequestMethod.GET)
    public InputStream processTheRequest( HttpServletRequest request, Model model) {
        return new FileInputStream( ...) ;
    }

}
```

演示

- **演示**
  - 基于简单的 Spring MVC 的应用程序

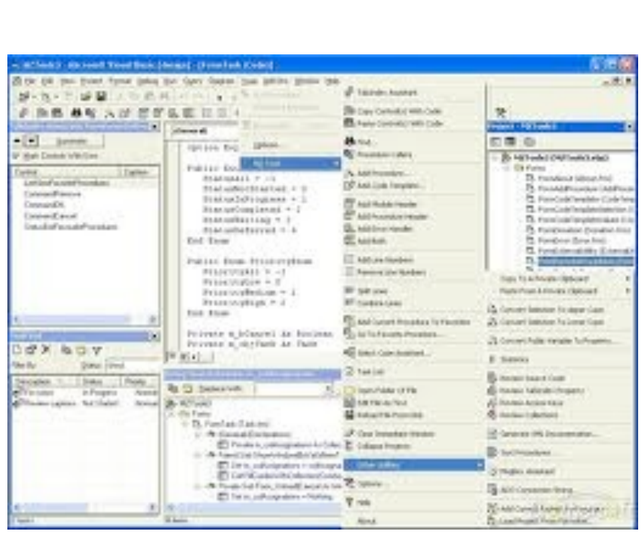
新热点...

简易终端首先控制了一切....

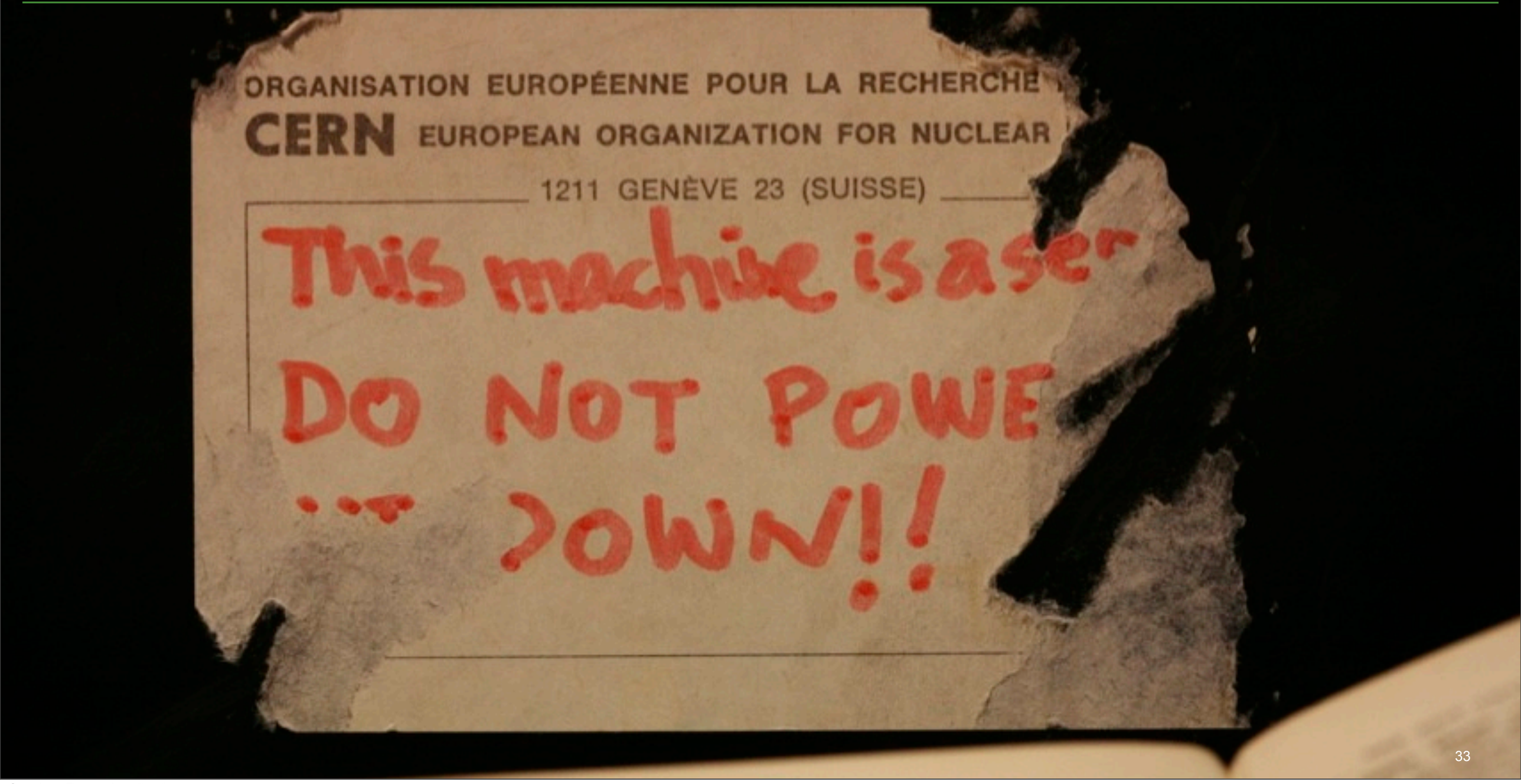


...之后有状态的应用程序开始占据主导地位...

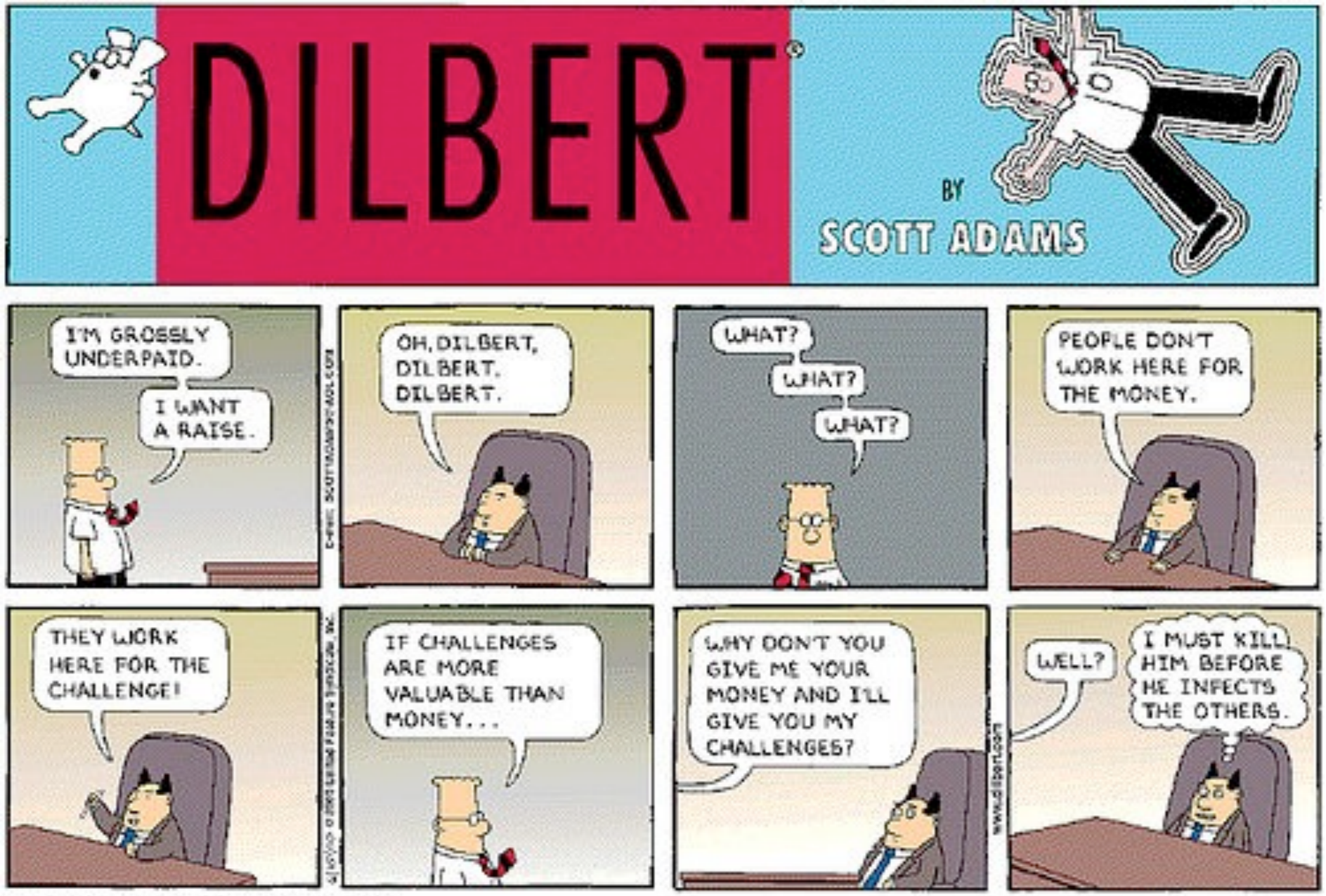
Microsoft®
Visual Basic®
for Applications



接着神奇的 Web 来了


ORGANISATION EUROPÉENNE POUR LA RECHERCHE
CERN EUROPEAN ORGANIZATION FOR NUCLEAR
1211 GENÈVE 23 (SUISSE)
This machine is a ser
DO NOT POWE
DOWN!!

但是 Web 不知道如何进行 UI 陈述... 所以我们开创了新方式...



....接着 Web 开始改进


HTML


BACKBONE.JS


ANGULARJS
by Google


jQuery
write less, do more.

JavaScript 究竟有多强大？ ...它开始了 Linux！！

bellard.org/jslinux/

```
Mem: 2876K used, 11192K free, 0K shrd, 192K buff, 1544K cached
CPU:  1.3% usr  0.6% sys  0.0% nic 93.9% idle  0.0% io  4.0% irq  0.0% sirq
Load average: 0.00 0.00 0.00 1/12 159
  PID  PPID USER     STAT   VSZ %MEM CPU %CPU COMMAND
  159   155 root     R     1144  8.1   0  4.6 top
  155     1 root     S     1160  8.2   0  0.0 sh
    1     0 root     S     1144  8.1   0  0.0 /bin/sh /sbin/init
    4     1 root     SW<      0  0.0   0  0.0 [khelper]
    2     1 root     SWN      0  0.0   0  0.0 [ksoftirqd/0]
    5     1 root     SW<      0  0.0   0  0.0 [kthread]
   20     5 root     SW<      0  0.0   0  0.0 [kblockd/0]
    3     1 root     SW<      0  0.0   0  0.0 [events/0]
   38     5 root     SW<      0  0.0   0  0.0 [kswapd0]
   39     5 root     SW<      0  0.0   0  0.0 [aio/0]
   36     5 root     SW       0  0.0   0  0.0 [pdflush]
   37     5 root     SW       0  0.0   0  0.0 [pdflush]
/var/root #
```

Clear clipboard

REST

Thin、Thick、Web、移动和 Rich 客户端：REST

- **起源**
  - 术语“表述性状态转移” 由 **Roy Fielding** 于 2000 年在他的博士论文中首次提出并定义
- **他的论文中提出了四种设计原理：**
  - 明确使用 HTTP 方法。
    - POST、GET、PUT、DELETE
    - CRUD 操作可以映射至这些现有方法
  - 无状态。
    - 状态依赖性限制或限制可扩展性
  - 公开 URI 等目录结构。
    - URI 应易于理解
  - 转移 XML 和/或 JavaScript Object Notation (JSON)。
    - 使用 XML 或 JSON 表示数据对象或属性

服务器上的 REST

- **Spring MVC 是 REST 支持的基础**
  - Spring 的服务器端 REST 支持基于标准控制器模型
- **JavaScript 和 HTML5 可以使用 JSON 数据有效负载**

客户端上的 REST

- **RestTemplate**
  - 提供十分简单且基于 REST 的惯用服务的使用
  - 也可以使用 Spring OXM。
  - Spring Integration 与 Spring Social 均在可实现的情况下基于 RestTemplate 构建。
- **Spring 不需配置便可支持多个转换器**
  - JAXB
  - JSON (Jackson)

为基于 REST 的服务构建客户端：RestTemplate

- Google 搜索示例

```
RestTemplate restTemplate = new RestTemplate();
String url = "https://ajax.googleapis.com/ajax/services/search/web?v=1.0&q={query}";
String result = restTemplate.getForObject(url, String.class, "SpringSource");
```

- 多个参数

```
RestTemplate restTemplate = new RestTemplate();
String url = "http://example.com/hotels/{hotel}/bookings/{booking}";
String result = restTemplate.getForObject(url, String.class, "42", “21”);
```

Thin、Thick、Web、移动和 Rich 客户端：RestTemplate

- **RestTemplate 类是客户端操作的核心**
  - 六种主要 HTTP 方法的入口点
    - DELETE - delete(...)
    - GET - getForObject(...)
    - HEAD - headForHeaders(...)
    - OPTIONS - optionsForAllow(...)
    - POST - postForLocation(...)
    - PUT - put(...)
    - 任何 HTTP 操作 - exchange(...) and execute(...)

Spring MVC 控制器的构造

Spring MVC 配置 – 配置

```
@Controller
public class CustomerController {

    @RequestMapping(value=”/url/of/my/resource”,
                    method = RequestMethod.GET)
    public @ResponseBody Customer processTheRequest(  ... ) {
       Customer c = service.getCustomerById( id) ;
       return c;
    }

    @Autowired CustomerService service;
}
```

Spring MVC 控制器的构造

Spring MVC 配置 – 配置

```
@Controller
public class CustomerController {

    @RequestMapping(value=”/url/of/my/someurl”,
                    method = RequestMethod.POST)
    public String processTheRequest( @RequestBody Customer postedCustomerObject) {
        // ...
        return “home”;
    }

}
```

POST http://127.0.0.1:8080/**url/of/my/someurl**

Spring MVC 控制器的构造

Spring MVC 配置 – 配置

```
@Controller
public class CustomerController {

    @RequestMapping(value=”/url/of/my/{id}”,
                    method = RequestMethod.POST)
    public String processTheRequest( @PathVariable(“id”) Long customerId,
                                     @RequestBody Customer postedCustomerObject) {
        // ...
        return “home”;
    }

}
```

POST http://127.0.0.1:8080/**url/of/my/someurl**

那么浏览器又如何呢？

- 问题：现代浏览器仅支持 GET 和 POST
- **解决方案：仅在** send _method 请求参数位于请求中时，使用 Spring 的 HiddenHttpMethodFilter

```
<filter>
      <filter-name>hiddenHttpMethodFilter</filter-name>
      <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>
   </filter>
   <filter-mapping>
      <filter-name>hiddenHttpMethodFilter</filter-name>
      <url-pattern>/</url-pattern>
      <servlet-name>appServlet</servlet-name>
   </filter-mapping>
```

演示

- **演示：**
- Spring REST 服务
- Spring REST 客户端

Spring 移动

Thin、Thick、Web、移动和 Rich 客户端：移动

- **最佳战略？本机开发**
  - 返回到客户端优化的 Web 应用程序
- **Spring MVC 3.1 特定于移动客户端的内容协商与呈现**
  - 对于其他设备
    - *(除了 Android，还有其他设备？)*

演示

- **演示：**
  - 使用特定于客户端的呈现的移动客户端

Spring Android

Thin、Thick、Web、移动和 Rich 客户端：移动

- **Spring REST 非常适合移动设备**
- **Spring MVC 3.1 特定于移动客户端的内容协商和呈现**
  - 针对其他设备
- **Spring Android**
  - RestTemplate

好的，那么.....


Does Spring Android
support dependency injection?
NO.
Y U NOO SUPPORT
DEPENDENCY INJECTION???
CHALLENGE ACCEPTED

Thin、Thick、Web、移动和 Rich 客户端：移动

CustomerServiceClient – 客户端

```
private <T> T extractResponse( ResponseEntity<T> response) {
      if (response != null && response().value() == 200) {
           return response.getBody();
      }
      throw new RuntimeException("couldn't extract response.");
}

@Override
public Customer updateCustomer(long id, String fn, String ln) {
      String urlForPath = urlForPath("customer/{customerId}");
      return  extractResponse(this.restTemplate.postForEntity(
            urlForPath, new Customer(id, fn, ln), Customer.class, id));
}
```

Thin、Thick、Web、移动和 Rich 客户端：移动

- **演示：**
  - 使用来自 Android 的 Spring REST 服务

使用 Spring Flex

Thin、Thick、Web、移动和 Rich 客户端：Flex

Spring Flex

- 十分简单以便将 Spring Bean 作为 Flex 服务公开
- 借助 **Adobe 的支持进行开发**
- 但其仍具有优势：
  - 表单驱动的应用程序
  - 视频、2D 和 3D 图形、声音
  - Adobe AIR
  - 十分快速的通信
  - 服务器端推送
- **Spring ActionScript** 是由 SpringSource“发起”的炫酷框架

Thin、Thick、Web、移动和 Rich 客户端：Flex

crm-flex-servlet.xml – server

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ...>

    <context:component-scan base-package="org.springsource.examples.sawt.web.flex"/>

    <mvc:default-servlet-handler/>

    <flex:message-broker mapping-order="1">
        <flex:mapping pattern="/messagebroker/*"/>
        <flex:message-service default-channels="my-streaming-amf,my-longpolling-amf,my-polling-amf"/>
    </flex:message-broker>

    <flex:remoting-destination ref="jdbcCustomerService" destination-id="customerService"/>

</beans>
```

CustomerForm.mxml – client

```
<fx:Declarations><s:RemoteObject id="cs" destination="customerService" endpoint="{serviceUrl}">
  <s:method name="createCustomer" result="create_resultHandler(event)"/>
  <s:method name="updateCustomer" result="update_resultHandler(event)"/>
 </s:RemoteObject> </fx:Declarations>
<fx:Script><![CDATA[  cs.updateCustomer( c.id, c.firstName, c.lastName); ]]>
</fx:Script>
```

演示

- 通过 BlazeDS 开放 Spring 服务
- 从 Flex 客户端使用

另一种方式

PhoneGap

- **始终使用 HTML5！**
  - 桌面浏览器界面上的 HTML5
  - 客户端上的 HTML5 + PhoneGap
  - 服务器端上基于 REST 的服务可将所有连接起来

社交通信

Spring Social

- 对 Spring 框架进行的扩展，与软件即服务提供商连接
- 功能...
  - 可扩展的连接框架
  - 连接控制器
  - Java API 绑定
  - 登录控制器
- http://www.springsource.org/spring-social

Spring Social 项目

- Spring Social Core
- Spring Social Facebook
- Spring Social Twitter
- Spring Social LinkedIn
- Spring Social TripIt
- Spring Social GitHub
- Spring Social Gowalla
- Spring Social Weibo https://github.com/liuce/spring-social-weibo
- Spring Social 示例
  - 包括 Showcase、Quickstart、Movies、Canvas、Twitter4J、Popup

Spring Social 的关键组件

- **连接工厂**
  - 创建连接；处理授权流的后端
- **连接控制器**
  - 协调基于 Web 的连接流
- **连接资源库**
  - 保持连接以长期使用
- **连接工厂定位器**
  - 供连接控制器和连接资源库使用以找到连接工厂
- **API 绑定**
  - 执行发送至 API 的请求， 绑定到域对象， 进行错误处理
- **提供商登录控制器**
  - 基于现有连接将用户登录到应用程序

将应用程序社交化的关键步骤

- **配置 Spring Social Bean**
  - 连接工厂定位器和连接工厂
  - 连接资源库
  - 连接控制器
  - API 绑定
- **创建连接状态视图**
- **注入/使用 API 绑定**

配置： ConnectionFactoryLocator

```
@Bean
@Scope(value="singleton", proxyMode=ScopedProxyMode.INTERFACES)
public ConnectionFactoryLocator connectionFactoryLocator() {
  ConnectionFactoryRegistry registry = new ConnectionFactoryRegistry();

  registry.addConnectionFactory(
      new TwitterConnectionFactory(
          environment.getProperty("twitter.consumerKey"),
          environment.getProperty("twitter.consumerSecret")));

  registry.addConnectionFactory(
      new FacebookConnectionFactory(
          environment.getProperty("facebook.clientId"),
          environment.getProperty("facebook.clientSecret")));

  return registry;
}
```

配置： ConnectionFactoryLocator

```
@Bean
@Scope(value="singleton", proxyMode=ScopedProxyMode.INTERFACES)
public ConnectionFactoryLocator connectionFactoryLocator() {
  ConnectionFactoryRegistry registry = new ConnectionFactoryRegistry();

  registry.addConnectionFactory(
      new TwitterConnectionFactory(
          environment.getProperty("twitter.consumerKey"),
          environment.getProperty("twitter.consumerSecret")));

  registry.addConnectionFactory(
      new FacebookConnectionFactory(
          environment.getProperty("facebook.clientId"),
          environment.getProperty("facebook.clientSecret")));

  return registry;
}
```

配置： ConnectionFactoryLocator

```
@Bean
@Scope(value="singleton", proxyMode=ScopedProxyMode.INTERFACES)
public ConnectionFactoryLocator connectionFactoryLocator() {
  ConnectionFactoryRegistry registry = new ConnectionFactoryRegistry();

  registry.addConnectionFactory(
      new TwitterConnectionFactory(
          environment.getProperty("twitter.consumerKey"),
          environment.getProperty("twitter.consumerSecret")));

  registry.addConnectionFactory(
      new FacebookConnectionFactory(
          environment.getProperty("facebook.clientId"),
          environment.getProperty("facebook.clientSecret")));

  return registry;
}
```

配置： ConnectionFactoryLocator

```
@Bean
@Scope(value="singleton", proxyMode=ScopedProxyMode.INTERFACES)
public ConnectionFactoryLocator connectionFactoryLocator() {
  ConnectionFactoryRegistry registry = new ConnectionFactoryRegistry();

  registry.addConnectionFactory(
      new TwitterConnectionFactory(
          environment.getProperty("twitter.consumerKey"),
          environment.getProperty("twitter.consumerSecret")));

  registry.addConnectionFactory(
      new FacebookConnectionFactory(
          environment.getProperty("facebook.clientId"),
          environment.getProperty("facebook.clientSecret")));

  return registry;
}
```

配置： ConnectionRepository

```
@Bean
@Scope(value="request", proxyMode=ScopedProxyMode.INTERFACES)
public ConnectionRepository connectionRepository() {
  Authentication authentication = SecurityContextHolder.getContext().
              getAuthentication();
  if (authentication == null) {
    throw new IllegalStateException(
        "Unable to get a ConnectionRepository: no user signed in");
  }
  return usersConnectionRepository().createConnectionRepository(
          authentication.getName());
}
```

配置： ConnectionRepository

```
@Bean
@Scope(value="request", proxyMode=ScopedProxyMode.INTERFACES)
public ConnectionRepository connectionRepository() {
  Authentication authentication = SecurityContextHolder.getContext().
              getAuthentication();
  if (authentication == null) {
    throw new IllegalStateException(
        "Unable to get a ConnectionRepository: no user signed in");
  }
  return usersConnectionRepository().createConnectionRepository(
          authentication.getName());
}


 @Bean
 @Scope(value="singleton", proxyMode=ScopedProxyMode.INTERFACES)
 public UsersConnectionRepository usersConnectionRepository() {
   return new JdbcUsersConnectionRepository(
              dataSource,
              connectionFactoryLocator(),
              Encryptors.noOpText());
 }
```

配置： ConnectController

```
@Bean
public ConnectController connectController() {
  return new ConnectController(connectionFactoryLocator(),
                               connectionRepository());
}
```

配置： ConnectController

```
@Bean
public ConnectController connectController() {
  return new ConnectController(connectionFactoryLocator(),
                               connectionRepository());
}
```

配置： ConnectController

```
@Bean
public ConnectController connectController() {
  return new ConnectController(connectionFactoryLocator(),
                  connectionRepository());
}
```

配置：API 绑定

```
@Bean
@Scope(value="request", proxyMode=ScopedProxyMode.INTERFACES)
public Facebook facebook() {
  Connection<Facebook> facebook =
          connectionRepository().findPrimaryConnection(Facebook.class);
  return facebook != null ? facebook.getApi() : new FacebookTemplate();
}
```

```
@Bean
@Scope(value="request", proxyMode=ScopedProxyMode.INTERFACES)
public Twitter twitter() {
  Connection<Twitter> twitter =
          connectionRepository().findPrimaryConnection(Twitter.class);
  return twitter != null ? twitter.getApi() : new TwitterTemplate();
}
```

配置：API 绑定

```
@Bean
@Scope(value="request", proxyMode=ScopedProxyMode.INTERFACES)
public Facebook facebook() {
  Connection<Facebook> facebook =
          connectionRepository().findPrimaryConnection(Facebook.class);
  return facebook != null ? facebook.getApi() : new FacebookTemplate();
}
```

```
@Bean
@Scope(value="request", proxyMode=ScopedProxyMode.INTERFACES)
public Twitter twitter() {
  Connection<Twitter> twitter =
          connectionRepository().findPrimaryConnection(Twitter.class);
  return twitter != null ? twitter.getApi() : new TwitterTemplate();
}
```

配置：API 绑定

```
@Bean
@Scope(value="request", proxyMode=ScopedProxyMode.INTERFACES)
public Facebook facebook() {
  Connection<Facebook> facebook =
        connectionRepository().findPrimaryConnection(Facebook.class);
  return facebook != null ? facebook.getApi() : new FacebookTemplate();
}
```

```
@Bean
@Scope(value="request", proxyMode=ScopedProxyMode.INTERFACES)
public Twitter twitter() {
  Connection<Twitter> twitter =
        connectionRepository().findPrimaryConnection(Twitter.class);
  return twitter != null ? twitter.getApi() : new TwitterTemplate();
}
```

配置：API 绑定

```
@Bean
@Scope(value="request", proxyMode=ScopedProxyMode.INTERFACES)
public Facebook facebook() {
  Connection<Facebook> facebook =
          connectionRepository().findPrimaryConnection(Facebook.class);
  return facebook != null ? facebook.getApi() : new FacebookTemplate();
}
```

```
@Bean
@Scope(value="request", proxyMode=ScopedProxyMode.INTERFACES)
public Twitter twitter() {
  Connection<Twitter> twitter =
          connectionRepository().findPrimaryConnection(Twitter.class);
  return twitter != null ? twitter.getApi() : new TwitterTemplate();
}
```

注入和使用 API 绑定

```
@Controller
public class TwitterTimelineController {

    private final Twitter twitter;

    @Inject
    public TwitterTimelineController(Twitter twitter) {
        this.twitter = twitter;
    }

    @RequestMapping(value="/twitter/tweet",
                    method=RequestMethod.POST)
    public String postTweet(String message) {
        twitter.timelineOperations().updateStatus(message);
        return "redirect:/twitter";
    }
}
```

注入和使用 API 绑定

```
@Controller
public class TwitterTimelineController {

    private final Twitter twitter;

    @Inject
    public TwitterTimelineController(Twitter twitter) {
        this.twitter = twitter;
    }

    @RequestMapping(value="/twitter/tweet",
                    method=RequestMethod.POST)
    public String postTweet(String message) {
        twitter.timelineOperations().updateStatus(message);
        return "redirect:/twitter";
    }
}
```

注入和使用 API 绑定

```
@Controller
public class TwitterTimelineController {

    private final Twitter twitter;

    @Inject
    public TwitterTimelineController(Twitter twitter) {
        this.twitter = twitter;
    }

    @RequestMapping(value="/twitter/tweet",
                    method=RequestMethod.POST)
    public String postTweet(String message) {
        twitter.timelineOperations().updateStatus(message);
        return "redirect:/twitter";
    }
}
```

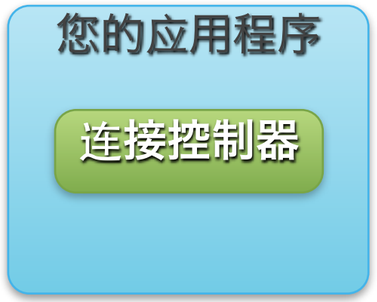
ConnectController 终端

- **GET /connect**
  - 显示所有提供商的连接状态
- **GET /connect/{provider}**
  - 显示给定提供商的连接状态
- **POST /connect/{provider}**
  - 开始授权流，重定向到提供商
- **GET /connect/{provider}?oauth_token={token}**
  - 处理 OAuth 1 回调
- **GET /connect/{provider}?code={authorization code}**
  - 处理 OAuth 2 回调
- **DELETE /connect/{provider}**
  - 将用户的所有连接移动至给定提供商
- **DELETE /connect/{provider}/{provider user ID}**
  - 将用户的特定连接移动至给定提供商

ConnectController 流

ConnectController 流

GET /connect/{provider ID}



服务提供商

显示连接状态页面

ConnectController 流

POST /connect/{provider ID}

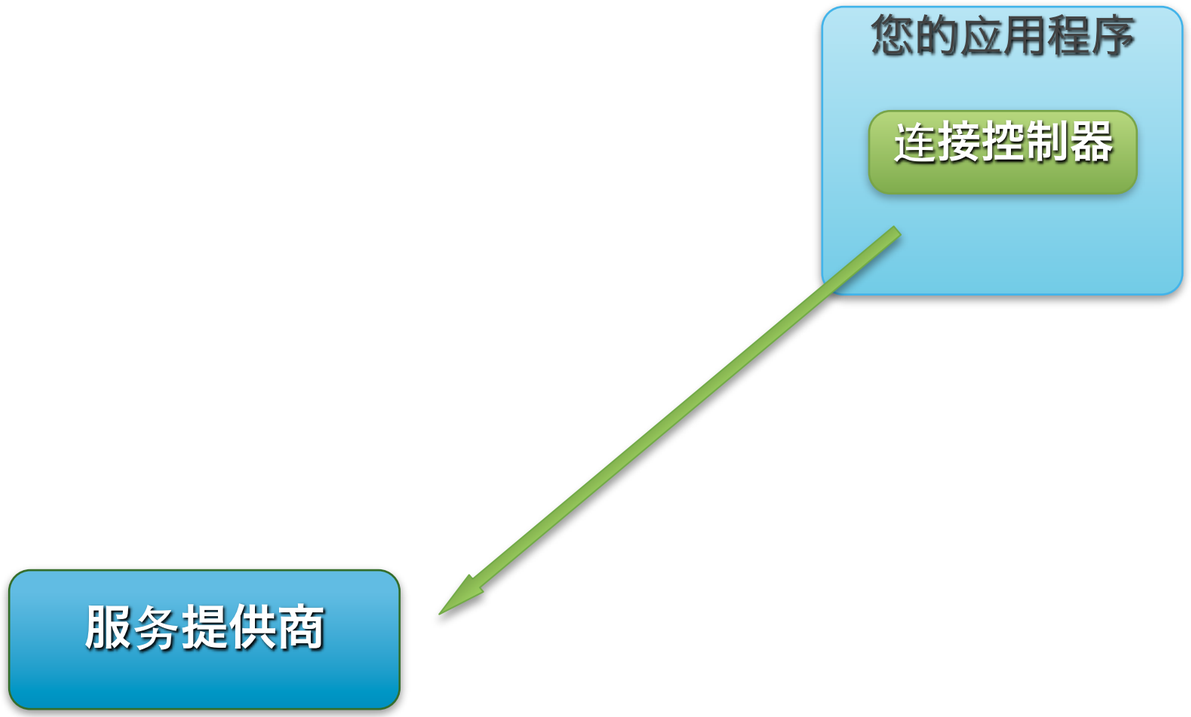
您的应用程序

连接控制器

服务提供商

开始连接流

ConnectController 流



提取请求令牌（仅 OAuth 1.0/1.0a）

ConnectController 流

您的应用程序

连接控制器

服务提供商

将浏览器重定向到提供商的授权页面

ConnectController 流

将浏览器重定向到提供商的授权页面

ConnectController 流

您的应用程序

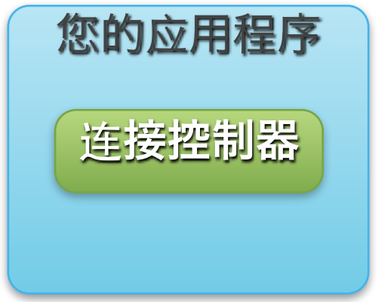
连接控制器

服务提供商

将浏览器重定向到提供商的授权页面

ConnectController 流

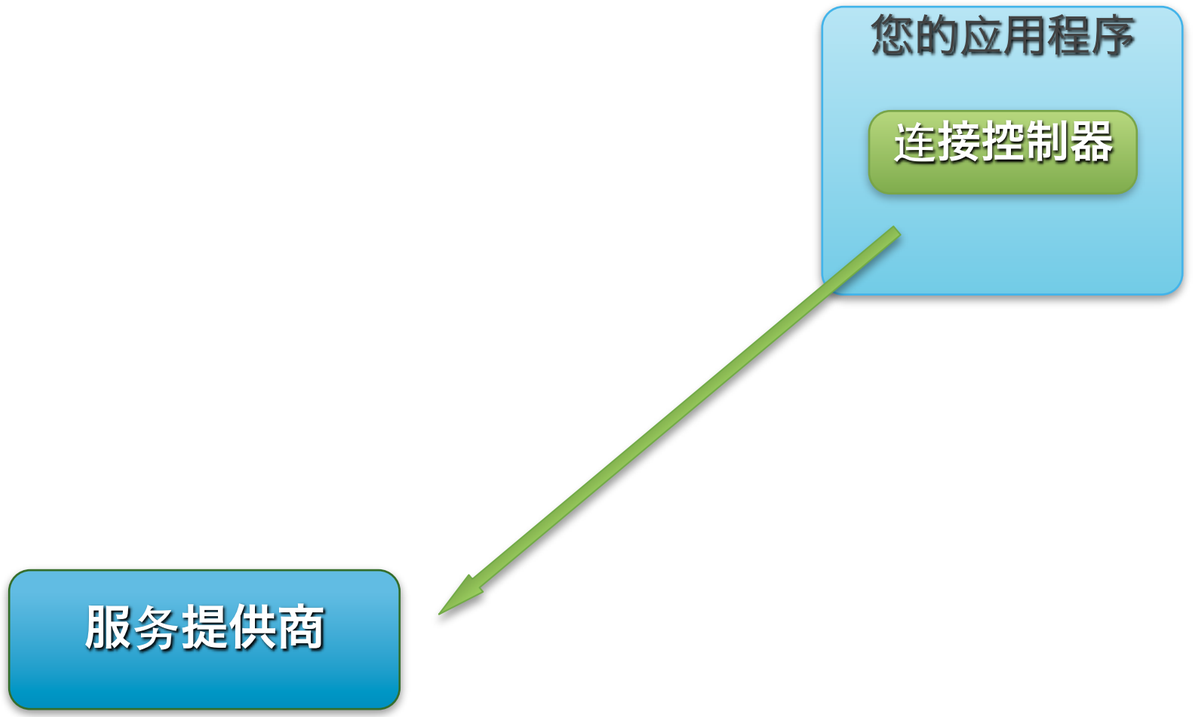
GET /connect/{provider ID}?oauth_token={token}
GET /connect/{provider ID}?code={code}



服务提供商

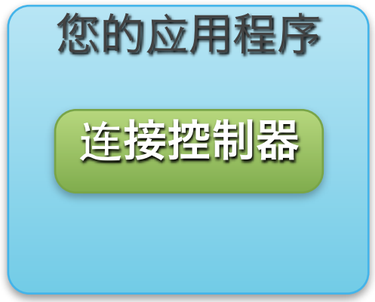
提供商重定向至回调 URL

ConnectController 流



使用请求令牌和/或代码交换访问令牌

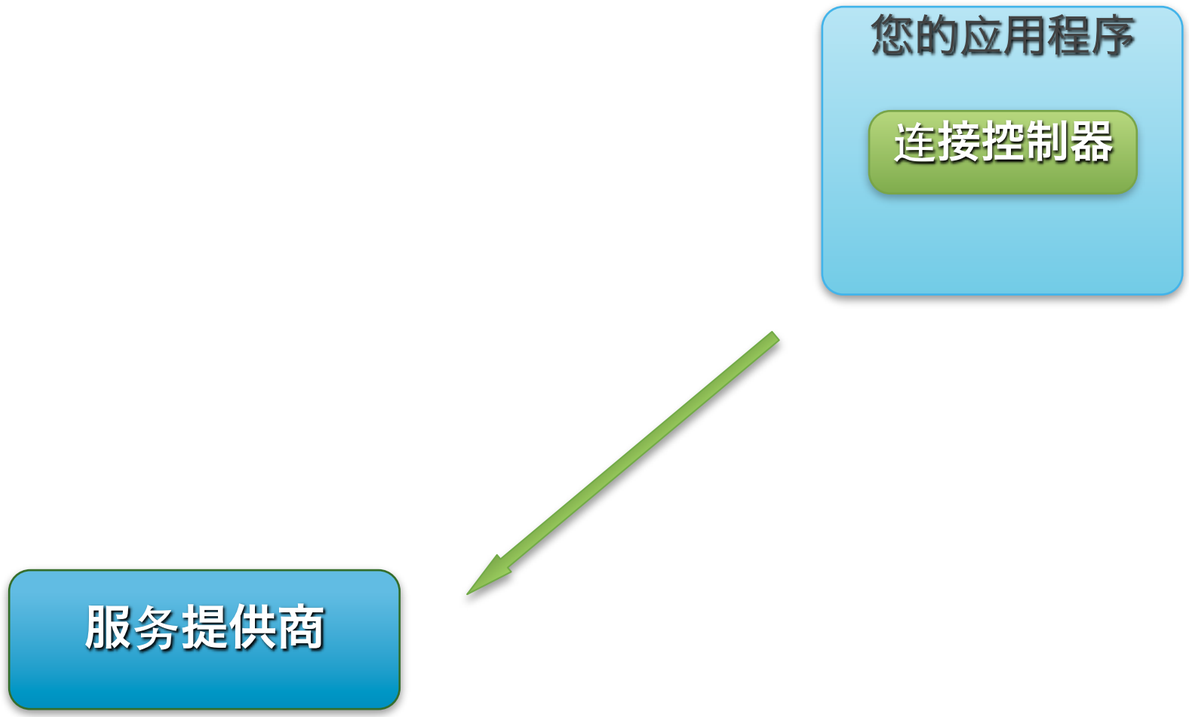
ConnectController 流



服务提供商

连接控制器存储连接资源库中的连接详细信息

ConnectController 流



应用程序可以通过 API 绑定进行 API 调用

连接状态页面视图

```
<form action="<c:url value="/connect/twitter" />" method="POST">
 <div class="formInfo">
  <p>
  You haven't created any connections with Twitter yet.
  Click the button to connect with your Twitter account.
  </p>
 </div>
 <p>
  <button type="submit">
  <img src="<c:url value="/resources/social/twitter/connect-with-twitter.png" />"/>
  </button>
 </p>
</form>
```

提供商登录

- 方便用户
- 支持使用应用程序连接作为凭据进行应用程序身份验证
- 使用提供商登录控制器进行实施
- 与任何提供商进行一致地合作

配置： ProviderSignInController

- 执行的流与连接控制器类似
- 比较连接（通过用户 ID）
- 如果存在匹配，则使用户登录至应用程序
- 否则，将用户发送至注册页面
  - 在注册后建立连接

```
@Bean
public ProviderSignInController providerSignInController(
            RequestCache requestCache) {
  return new ProviderSignInController(connectionFactoryLocator(),
                              usersConnectionRepository(),
                              new SimpleSignInAdapter(requestCache));
}
```

配置： ProviderSignInController

- 执行的流与连接控制器类似
- 比较连接（通过用户 ID）
- 如果存在匹配，则使用户登录至应用程序
- 否则，将用户发送至注册页面
  - 在注册后建立连接

```
@Bean
public ProviderSignInController providerSignInController(
            RequestCache requestCache) {
  return new ProviderSignInController(connectionFactoryLocator(),
                        usersConnectionRepository(),
                        new SimpleSignInAdapter(requestCache));
}
```

配置： ProviderSignInController

- 执行的流与连接控制器类似
- 比较连接（通过用户 ID）
- 如果存在匹配，则使用户登录至应用程序
- 否则，将用户发送至注册页面
  - 在注册后建立连接

```
@Bean
public ProviderSignInController providerSignInController(
            RequestCache requestCache) {
  return new ProviderSignInController(connectionFactoryLocator(),
                        usersConnectionRepository(),
                         new SimpleSignInAdapter(requestCache));
}
```

配置： ProviderSignInController

- 执行的流与连接控制器类似
- 比较连接（通过用户 ID）
- 如果存在匹配，则使用户登录至应用程序
- 否则，将用户发送至注册页面
  - 在注册后建立连接

```
@Bean
public ProviderSignInController providerSignInController(
            RequestCache requestCache) {
  return new ProviderSignInController(connectionFactoryLocator(),
                              usersConnectionRepository(),
                              new SimpleSignInAdapter(requestCache));
}
```

ProviderSignInController 终端

- **POST /signin/{provider}**
  - 开始授权流，重定向到提供商
- **GET /signin/{provider}?oauth_token={token}**
  - 处理 OAuth 1 回调
- **GET /signin/{provider}?code={authorization code}**
  - 处理 OAuth 2 回调
- **GET /signin**
  - 在未发送 oauth 令牌或代码时处理回调
  - 可能表示用户拒绝授权

Spring Security OAuth

Spring Security OAuth

- **对 Spring Security 进行的扩展**
  - 原先为社区贡献（现在正式为项目的一部分）
- **功能...**
  - OAuth 服务类型的终端管理
  - （以及相应的客户端管理）
  - 令牌管理（持久性，授权）
  - 针对 Web 并通过标准 Spring Security 集成
- **springsource.org/spring-security/oauth**

使用 Spring MVC 设置 Spring Security...

```
<http access-denied-page="/login.jsp"
    xmlns="http://www.springframework.org/schema/security">

        <form-login authentication-failure-url="/login.jsp"
              default-target-url="/index.jsp"
              login-page="/login.jsp"
              login-processing-url="/login.do" />

        <logout logout-success-url="/index.jsp" logout-url="/logout.do" />

        <anonymous />

        <intercept-url pattern="/sparklr/**" access="ROLE_USER" />

        <custom-filter ref="oauth2ClientFilter" after="EXCEPTION_TRANSLATION_FILTER" />

</http>
```

然后将我们的 OAuth 终端告知 Spring Security

```
<authentication-manager
    xmlns="http://www.springframework.org/schema/security">
        <authentication-provider>
                <user-service>
                <user name="marissa" password="wombat"
                        authorities="ROLE_USER" />
                <user name="sam" password="kangaroo"
                        authorities="ROLE_USER" />
                </user-service>
        </authentication-provider>
</authentication-manager>

<oauth:client id="oauth2ClientFilter"
    redirect-on-error="${redirectOnError:false}" />

<oauth:resource id="sparklr"
    type="authorization_code" client-id="tonr"
    client-secret="secret" access-token-uri="${accessTokenUri}"
    user-authorization-uri="${userAuthorizationUri}" scope="read" />
```

接着为客户端获取一个 RestTemplate 实例...

```
<bean class="org.springframework.security.oauth2.client.OAuth2RestTemplate">
  <constructor-arg ref = “sparklr”/>
</bean>
```

并为 REST 剩余部分获取

- **Spring Data REST –** 公开基于 Spring Data 资源库构建的（JPA、MongoDB、GemFire、Neo4J、Redis）资源库（**请阅读我的朋友 Sergi 的演讲！**）
- **Spring HATEOAS -** 借助对将 HTTP 作为应用程序状态引擎的支持，将您的 REST 带入下一阶段

@starbuxman | josh.long@springsource.com

http://slideshare.net/joshlong

问题？