- .Net
- 移动
- Android
- iOS
- HTML 5
- JavaScript
- 函数式编程
- Web API

特别专题 语言 & 开发

京东618：商城分布式智能容器DNS实践


随着京东业务的高速增长，以及JDOS2.0的线上大规模运营，进而容器集群的编排成为常态，Pod失效也成为常态，RS（Replication Set）在处理失效Pod时候会带来IP的变化。这样容器之间基于IP相互访问就有可能存在问题。所以一个强大的能支持百万级hostname域名解析服务，可以很好地解决这个问题。

浏览所有 **语言 & 开发**

- 架构 & 设计
  - 架构
  - 企业架构
  - 性能和可伸缩性
  - 设计
  - 案例分析
  - 设计模式
  - 安全

特别专题 架构 & 设计

京东618：商城分布式智能容器DNS实践


随着京东业务的高速增长，以及JDOS2.0的线上大规模运营，进而容器集群的编排成为常态，Pod失效也成为常态，RS（Replication Set）在处理失效Pod时候会带来IP的变化。这样容器之间基于IP相互访问就有可能存在问题。所以一个强大的能支持百万级hostname域名解析服务，可以很好地解决这个问题。

浏览所有 **架构 & 设计**

- 数据科学
  - 大数据
  - NoSQL
  - 数据库

特别专题 数据科学

架构师（2017年6月）

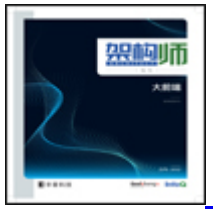
本期主要内容：“从此社区再无 Docker？”那 “Moby” 又是什么？Kotlin成为正式的Android编程语言；如何构建一套高可用的移动消息推送平台？阿里巴巴为什么要选择星际争霸作为AI算法研究环境？人工智能与软件架构

浏览所有 **数据科学**

- 文化 & 方法
  - Agile
  - 领导能力
  - 团队协作
  - 测试
  - 用户体验
  - Scrum
  - 精益

特别专题 文化 & 方法

架构师特刊：大前端



本期主要内容：当我们在谈大前端的时候， 我们谈的是什么；如何落地和管理一个“大前端”团队?饿了么大前端团 队解密；2017，我们来聊聊 Node.js；Conversational UI：AI时代的 人机交互模式

浏览所有 **文化 & 方法**

- DevOps
  - 持续交付
  - 自动化操作
  - 云计算

特别专题 DevOps

无服务器架构将DevOps带入新层次


无服务器架构不仅补充了DevOps的理念，更改进了当前IT组织实现更高业务敏捷性的观念。它致力于快速交付商业价值并持续改进和学习，这极有可能会带来文化上大范围的改变，甚至对那些已采用了DevOps文化和实践的组织也不例外。

浏览所有 **DevOps**

Java深度历险（六）——Java注解

| 作者 成富 发布于 2011年3月23日. 估计阅读时间: 2 分钟 |

在开发Java程序，尤其是Java EE应用的时候，总是免不了与各种配置文件打交道。以Java EE中典型的S(pring)S(truts)H(ibernate)架构来说，Spring、Struts和Hibernate这三个框架都有自己的XML格式的配置文件。这些配置文件需要与Java源代码保存同步，否则的话就可能出现错误。而且这些错误有可能到了运行时刻才被发现。把同一份信息保存在两个地方，总是个坏的主意。理想的情况是在一个地方维护这些信息就好了。其它部分所需的信息则通过自动的方式来生成。JDK 5中引入了源代码中的注解（annotation）这一机制。注解使得Java源代码中不但可以包含功能性的实现代码，

还可以添加元数据。注解的功能类似于代码中的注释，所不同的是注解不是提供代码功能的说明，而是实现程序功能的重要组成部分。Java注解已经在很多框架中得到了广泛的使用，用来简化程序中的配置。

使用注解

在一般的Java开发中，最常接触到的可能就是@Override和@SupressWarnings这两个注解了。使用@Override的时候只需要一个简单的声明即可。这种称为标记注解（marker annotation），它的出现就代表了某种配置语义。而其它的注解是可以有自己的配置参数的。配置参数以名值对的方式出现。使用 @SupressWarnings的时候需要类似@SupressWarnings({"uncheck", "unused"})这样的语法。在括号里面的是该注解可供配置的值。由于这个注解只有一个配置参数，该参数的名称默认为value，并且可以省略。而花括号则表示是数组类型。在JPA中的@Table注解使用类似@Table(name = "Customer", schema = "APP")这样的语法。从这里可以看到名值对的用法。在使用注解时候的配置参数的值必须是编译时刻的常量。

从某种角度来说，可以把注解看成是一个XML元素，该元素可以有不同的预定义的属性。而属性的值是可以在声明该元素的时候自行指定的。在代码中使用注解，就相当于把一部分元数据从XML文件移到了代码本身之中，在一个地方管理和维护。

开发注解

在一般的开发中，只需要通过阅读相关的API文档来了解每个注解的配置参数的含义，并在代码中正确使用即可。在有些情况下，可能会需要开发自己的注解。这在库的开发中比较常见。注解的定义有点类似接口。下面的代码给出了一个简单的描述代码分工安排的注解。通过该注解可以在源代码中记录每个类或接口的分工和进度情况。

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface Assignment {
   String assignee();
   int effort();
   double finished() default 0;
}
```

@interface用来声明一个注解，其中的每一个方法实际上是声明了一个配置参数。方法的名称就是参数的名称，返回值类型就是参数的类型。可以通过default来声明参数的默认值。在这里可以看到@Retention和@Target这样的元注解，用来声明注解本身的行为。@Retention用来声明注解的保留策略，有CLASS、RUNTIME和SOURCE这三种，分别表示注解保存在类文件、JVM运行时刻和源代码中。只有当声明为RUNTIME的时候，才能够在运行时刻通过反射API来获取到注解的信息。@Target用来声明注解可以被添加在哪些类型的元素上，如类型、方法和域等。

处理注解

在程序中添加的注解，可以在编译时刻或是运行时刻来进行处理。在编译时刻处理的时候，是分成多趟来进行的。如果在某趟处理中产生了新的Java源文件，那么就需要另外一趟处理来处理新生成的源文件。如此往复，直到没有新文件被生成为止。在完成处理之后，再对Java代码进行编译。JDK 5中提供了apt工具用来对注解进行处理。apt是一个命令行工具，与之配套的还有一套用来描述程序语义结构的Mirror API。Mirror API（com.sun.mirror.*）描述的是程序在编译时刻的静态结构。通过Mirror API可以获取到被注解的Java类型元素的信息，从而提供相应的处理逻辑。具体的处理工作交给apt工具来完成。编写注解处理器的核心是AnnotationProcessorFactory和AnnotationProcessor两个接口。后者表示的是注解处理器，而前者则是为某些注解类型创建注解处理器的工厂。

以上面的注解Assignment为例，当每个开发人员都在源代码中更新进度的话，就可以通过一个注解处理器来生成一个项目整体进度的报告。 首先是注解处理器工厂的实现。

```
public class AssignmentApf implements AnnotationProcessorFactory {
   public AnnotationProcessor getProcessorFor(Set<AnnotationTypeDeclaration> atds,? AnnotationProcessorEnvironment env) {
      if (atds.isEmpty()) {
        return AnnotationProcessors.NO_OP;
      }
      return new AssignmentAp(env); //返回注解处理器
   }
   public Collection<String> supportedAnnotationTypes() {
```

```
        return Collections.unmodifiableList(Arrays.asList("annotation.Assignment"));
    }
    public Collection<String> supportedOptions() {
        return Collections.emptySet();
    }
}
```

AnnotationProcessorFactory接口有三个方法：getProcessorFor是根据注解的类型来返回特定的注解处理器；supportedAnnotationTypes是返回该工厂生成的注解处理器所能支持的注解类型；supportedOptions用来表示所支持的附加选项。在运行apt命令行工具的时候，可以通过-A来传递额外的参数给注解处理器，如-Averbose=true。当工厂通过 supportedOptions方法声明了所能识别的附加选项之后，注解处理器就可以在运行时刻通过AnnotationProcessorEnvironment的getOptions方法获取到选项的实际值。注解处理器本身的基本实现如下所示。

```
public class AssignmentAp implements AnnotationProcessor {
    private AnnotationProcessorEnvironment env;
    private AnnotationTypeDeclaration assignmentDeclaration;
    public AssignmentAp(AnnotationProcessorEnvironment env) {
        this.env = env;
        assignmentDeclaration = (AnnotationTypeDeclaration) env.getTypeDeclaration("annotation.Assignment");
    }
    public void process() {
        Collection<Declaration> declarations = env.getDeclarationsAnnotatedWith(assignmentDeclaration);
        for (Declaration declaration : declarations) {
            processAssignmentAnnotations(declaration);
        }
    }
    private void processAssignmentAnnotations(Declaration declaration) {
        Collection<AnnotationMirror> annotations = declaration.getAnnotationMirrors();
        for (AnnotationMirror mirror : annotations) {
            if (mirror.getAnnotationType().getDeclaration().equals(assignmentDeclaration)) {
                Map<AnnotationTypeElementDeclaration, AnnotationValue> values = mirror.getElementValues();
                String assignee = (String) getAnnotationValue(values, "assignee"); //获取注解的值
            }
        }
    }
}
```

注解处理器的处理逻辑都在process方法中完成。通过一个声明（Declaration）的getAnnotationMirrors方法就可以获取到该声明上所添加的注解的实际值。得到这些值之后，处理起来就不难了。

在创建好注解处理器之后，就可以通过apt命令行工具来对源代码中的注解进行处理。 命令的运行格式是apt -classpath bin -factory annotation.apt.AssignmentApf src/annotation/work/*.java，即通过-factory来指定注解处理器工厂类的名称。实际上，apt工具在完成处理之后，会自动调用javac来编译处理完成后的源代码。

JDK 5中的apt工具的不足之处在于它是Oracle提供的私有实现。在JDK 6中，通过JSR 269把自定义注解处理器这一功能进行了规范化，有了新的javax.annotation.processing这个新的API。对Mirror API也进行了更新，形成了新的javax.lang.model包。注解处理器的使用也进行了简化，不需要再单独运行apt这样的命令行工具，Java编译器本身就可以完成对注解的处理。对于同样的功能，如果用JSR 269的做法，只需要一个类就可以了。

```
@SupportedSourceVersion(SourceVersion.RELEASE_6)
@SupportedAnnotationTypes("annotation.Assignment")
public class AssignmentProcess extends AbstractProcessor {
   private TypeElement assignmentElement;
   public synchronized void init(ProcessingEnvironment processingEnv) {
      super.init(processingEnv);
      Elements elementUtils = processingEnv.getElementUtils();
      assignmentElement = elementUtils.getTypeElement("annotation.Assignment");
   }
   public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment roundEnv) {
      Set<? extends Element> elements = roundEnv.getElementsAnnotatedWith(assignmentElement);
      for (Element element : elements) {
         processAssignment(element);
      }
   }
   private void processAssignment(Element element) {
      List<? extends AnnotationMirror> annotations = element.getAnnotationMirrors();
      for (AnnotationMirror mirror : annotations) {
         if (mirror.getAnnotationType().asElement().equals(assignmentElement)) {
            Map<? extends ExecutableElement, ? extends AnnotationValue> values = mirror.getElementValues();
            String assignee = (String) getAnnotationValue(values, "assignee"); //获取注解的值
         }
      }
   }
}
```

仔细比较上面两段代码，可以发现它们的基本结构是类似的。不同之处在于JDK 6中通过元注解@SupportedAnnotationTypes来声明所支持的注解类型。另外描述程序静态结构的javax.lang.model包使用了不同的类型名称。使用的时候也更加简单，只需要通过javac -processor annotation.pap.AssignmentProcess Demo1.java这样的方式即可。

上面介绍的这两种做法都是在编译时刻进行处理的。而有些时候则需要在运行时刻来完成对注解的处理。这个时候就需要用到Java的反射API。反射API提供了在运行时刻读取注解信息的支持。不过前提是注解的保留策略声明的是运行时。Java反射API的AnnotatedElement接口提供了获取类、方法和域上的注解的实用方法。比如获取到一个Class类对象之后，通过getAnnotation方法就可以获取到该类上添加的指定注解类型的注解。

实例分析

下面通过一个具体的实例来分析说明在实践中如何来使用和处理注解。假定有一个公司的雇员信息系统，从访问控制的角度出发，对雇员的工资的更新只能由具有特定角色的用户才能完成。考虑到访问控制需求的普遍性，可以定义一个注解来让开发人员方便的在代码中声明访问控制权限。

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface RequiredRoles {
   String[] value();
}
```

下一步则是如何对注解进行处理，这里使用的Java的反射API并结合动态代理。下面是动态代理中的InvocationHandler接口的实现。

```
public class AccessInvocationHandler<T> implements InvocationHandler {
   final T accessObj;
   public AccessInvocationHandler(T accessObj) {
      this.accessObj = accessObj;
   }
   public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
      RequiredRoles annotation = method.getAnnotation(RequiredRoles.class); //通过反射API获取注解
      if (annotation != null) {
         String[] roles = annotation.value();
         String role = AccessControl.getCurrentRole();
         if (!Arrays.asList(roles).contains(role)) {
            throw new AccessControlException("The user is not allowed to invoke this method.");
         }
      }
      return method.invoke(accessObj, args);
   }
}
```

在具体使用的时候，首先要通过Proxy.newProxyInstance方法创建一个EmployeeGateway的接口的代理类，使用该代理类来完成实际的操作。

参考资料

- JDK 5和JDK 6中的apt工具说明文档
- Pluggable Annotation Processing API

- APT: Compile-Time Annotation Processing with Java

感谢张凯峰对本文的策划和审校。

- 领域
- **语言 & 开发**
- **架构 & 设计**
- **文化 & 方法**
- 专栏
- 代码分析
- Java深度历险
- 专栏
- 调试
- Java
- 测试

相关内容

Java 9正式版有可能被推迟到9月21号发布

AWS再迎大师加盟：Java之父James Gosling决定效力

JCP EC投票反对Java平台模块系统

Azul Systems推出Falcon，一个基于LLVM的新的Java即时编译器

IBM和Red Hat会对Java模块系统（Jigsaw）投反对票

您好，朋友！

您需要 注册一个InfoQ账号 或者 登录 才能进行评论。在您完成注册后还需要进行一些设置。

获得来自InfoQ的更多体验。

告诉我们您的想法

允许的HTML标签: a,b,br,blockquote,i,li,pre,u,ul,p

当有人回复此评论时请E-mail通知我

发送信息

社区评论

代码示例挺清晰的 by 张 龙 Posted
Re: 代码示例挺清晰的 by jacle larry Posted
Re: 代码示例挺清晰的 by 版纳瑞 Binary Posted
Re: 代码示例挺清晰的 by cheung roc Posted
Re: 代码示例挺清晰的 by zhiguo wang Posted
学习了 by Qu Goddamned Posted
Re: 学习了 by i reid Posted
Re: 学习了 by 霍 泰稳 Posted

annotation使用三步曲 by hu kensun Posted
class AssignmentAp 少了getAnnotationValue方法。。 by 叶 紫萱 Posted
Re: class AssignmentAp 少了getAnnotationValue方法。。 by Huang Joe Posted
获取角色这里疑问 by Mooner guo Posted

代码示例挺清晰的 by "张 龙"

rt，不过这算是java开发者都应该掌握的基础知识了吧

- 喜欢
- 回复
- 回到顶部

Re: 代码示例挺清晰的 by "jacle larry"

最基础的东西 往往容易忽视

- 喜欢
- 回复
- 回到顶部

学习了 by "Qu Goddamned"

通过这篇文章，终于理解了lombok的实现机理，谢谢。
如果文章给出的是能运行的示例，而不仅仅是代码片段就好了。

- 喜欢
- 回复
- 回到顶部

Re: 代码示例挺清晰的 by "版纳瑞 Binary"

> rt，不过这算是java开发者都应该掌握的基础知识了吧

惭愧啊，java了好几年了，还不知道这些知识，看来得多充充电了

- 喜欢
- 回复
- 回到顶部

annotation使用三步曲 by "hu kensun"

annotation使用三步曲,
1.define annotation interface
2.define parser annotation class
3.use annotation and it's parser
在我framework中就用上了,做配置真好真方便。

- 喜欢
- 回复
- 回到顶部

Re: 代码示例挺清晰的 by "cheung roc"

唉，一样啊！

- 喜欢
- 回复
- 回到顶部

Re: 代码示例挺清晰的 by "zhiguo wang"

学习了，非常感谢~

- 喜欢
- 回复
- 回到顶部

class AssignmentAp 少了getAnnotationValue方法。。 by "叶 紫萱"

class AssignmentAp 少了getAnnotationValue方法。。

- 喜欢
- 回复
- 回到顶部

Re: class AssignmentAp 少了getAnnotationValue方法。。 by "Huang Joe"

确实如此,希望作者能补一下.我自己写了一个,做个参考吧:

```
private Object getAnnotationValue(Map<? extends ExecutableElement, ? extends AnnotationValue> values, String annotationFiledName) {
for (Entry<? extends ExecutableElement, ? extends AnnotationValue> entry : values.entrySet()) {
if (entry.getKey().getSimpleName().contentEquals(annotationFiledName)) {
return entry.getValue().getValue();
}
}
return null;
}</?></?>
```

- 喜欢
- 回复
- 回到顶部

Re: 学习了 by "i reid"

其实我也想说这句话，写东西写不全，让看的人还要各处去找答案，懂但不一定会写，我觉得。

- 喜欢
- 回复
- 回到顶部

Re: 学习了 by "霍 泰稳"

你悟道了，说是一方面，写是另一方面，能说的不一定能写。这也是为什么有些朋友愿意去做演讲，但不愿意写文章。写文章要字斟句酌，消耗的能量更多。

- 喜欢
- 回复

- 回到顶部

获取角色这里疑问 by "Mooner guo"

能评论吗？请问AccessControl.getCurrentRole()这个是ThreadLocal做的吧？

- 喜欢
- 回复
- 回到顶部

关闭

by

发布于

- 查看
- 回复
- 回到顶部

关闭

引用原消息

主题 您的回复

允许的HTML标签: a,b,br,blockquote,i,li,pre,u,ul,p

当有人回复此评论时请E-mail通知我

发送信息 取消

关闭

主题 您的回复

允许的HTML标签: a,b,br,blockquote,i,li,pre,u,ul,p

当有人回复此评论时请E-mail通知我

取消

关闭

OK

12 讨论

赞助商链接

相关内容

- Java 9正式版有可能被推迟到9月21号发布 2017年6月5日
- AWS再迎大师加盟：Java之父James Gosling决定效力 2017年5月23日

- Invokedynamic：Java的秘密武器 2017年3月24日

- 测试RxJava2 2017年3月15日

- 阿里巴巴Java开发手册 2017年3月7日

- InfoQ观点：Java EE的未来 2017年3月6日

- RxJava2实例解析 2017年2月28日

- RxJava2类库解析 2017年2月28日

- Java探针技术在应用安全领域的新突破 2017年1月26日

- JVM 虚拟化—— 重新定义 Java 容器热部署资源管理机制 2017年1月19日

- Java 9、OSGi以及模块化的未来（第二部分） 2017年1月18日

- Java 模块化技术演进和对现有应用微服务化的意义 2017年1月12日

赞助商内容

- 智能云时代 百度云存储技术如何构建及解决方案

百度云基于长期的存储技术积累，在云上推出了对象存储，块存储等多种存储产品和解决方案，来满足ABC时代用户的不同场景需求。本次沙龙，百度云产品经理、资深工程师和高级架构师将全面的解析百度云存储系统和解决方案，帮助开发者更加清晰了解云存储技术的实现过程及其应用之道！

- 如何将机器学习落地到实际开发？

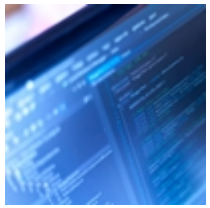
  我们必须将深度学习如何与高性能实时处理框架结合在一起，以解决现实问题，而不仅仅局限于一个理想实验

相关内容

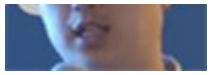
- 数据中心操作人员：艰难地在针对VM构建的基础设施上运行容器 2017年5月26日
- Kotlin成为正式的Android编程语言 2017年5月23日

- 想知道垃圾回收暂停的过程中发生了什么吗？查查垃圾回收日志就知道了！ 2017年5月17日

- JVM上的确定性执行机制 2017年5月12日

- NASA的10条代码编写原则 2017年5月2日 
- 使用模板将Web服务的结果转换为标记语言 2017年3月23日 
- 多形态MVC式Web架构：完成实时响应 2017年3月22日 
- 拥抱可变性 2017年3月8日 
- 书评：实战Apache JMeter 2017年2月6日 

- “理解数据科学”系列文章 2017年1月25日 


- 浅谈代码复用攻击与防御 2017年1月9日 

赞助商内容

- 百度技术沙龙免费报名，听大牛讲解云存储技术及应用之道

本次沙龙，百度云的存储产品经理和架构师将系统的介绍百度云的存储产品和解决方案，以及背后的系统架构演进和关键技术实践。

相关内容

- 观点：完美代码只是一个幻想 2017年1月3日 
- NetflixOSS：Hollow正式发布 2016年12月30日 
- 京东618：机器学习与商品数据挖掘和知识抽取 2017年6月18日
- 京东618：商城交易平台的高可用架构之路 2017年6月18日
- 京东618：智能机器人JIMI的进击之路 2017年6月18日
- 京东618：六年历程步步为营，京东商城的安全保卫战 2017年6月18日
- 京东618：商城分布式智能容器DNS实践 2017年6月18日 
- 京东618：一个中心五个原则，谈谈物流系统的大促优化实践 2017年6月18日 
- 京东618：容器技法日趋娴熟，数个项目即将开源回馈社区 2017年6月18日 
- 京东618：ReactNative框架在京东无线端的实践 2017年6月18日 

InfoQ每周精要

订阅InfoQ每周精要，加入拥有25万多名资深开发者的庞大技术社区。 

…升级全链路压测方案，打造军演机器人ForceBot 2017年6月18日