创建型模式

(原型模式、单例模式)

蔡茂华（包子）

AsiaInfo 亚信

原型模式

- 原型模式
  - ✓ 模式动机与定义
  - ✓ 模式结构与分析
  - ✓ 模式实例与解析
  - ✓ 模式效果与应用
  - ✓ 模式扩展

原型模式

- 模式动机

原型模式

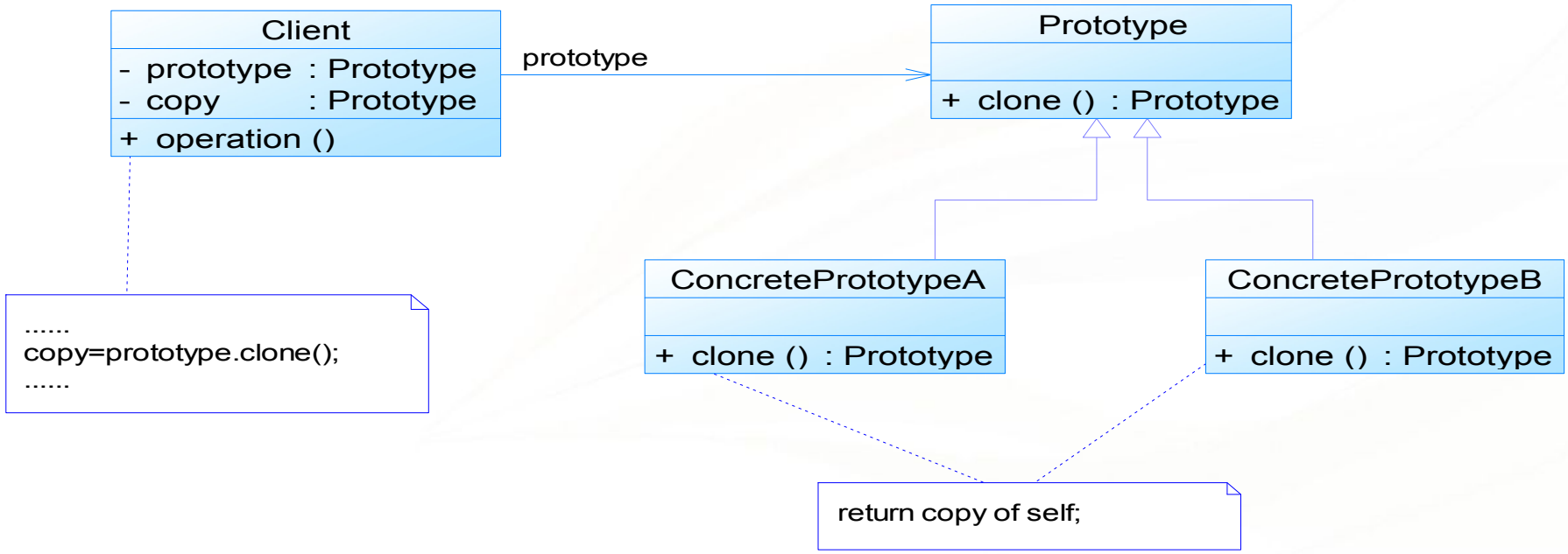
- 模式动机
  - ✓ 在面向对象系统中，使用原型模式来复制一个对象自身，从而克隆出多个与原型对象一模一样的对象。
  - ✓ 在软件系统中，有些对象的创建过程较为复杂，而且有时候需要频繁创建，原型模式通过给出一个原型对象来指明所要创建的对象的类型，然后用复制这个原型对象的办法创建出更多同类型的对象，这就是原型模式的意图所在。

原型模式

- 模式定义
  - ✓ 原型模式(Prototype Pattern)：原型模式是一种对象创建型模式，用原型实例指定创建对象的种类，并且通过复制这些原型创建新的对象。原型模式允许一个对象再创建另外一个可定制的对象，无须知道任何创建的细节。
  - ✓ 原型模式的基本工作原理是通过将一个原型对象传给那个要发动创建的对象，这个要发动创建的对象通过请求原型对象拷贝原型自己来实现创建过程。

原型模式

- 模式结构


Client
- prototype : Prototype
- copy : Prototype
+ operation ()
prototype
Prototype
+ clone () : Prototype
ConcretePrototypeA
+ clone () : Prototype
ConcretePrototypeB
+ clone () : Prototype
.....
copy=prototype.clone();
.....
return copy of self;

原型模式

- 模式结构
  - ✓ **原型模式包含如下角色：**
    - Prototype：抽象原型类
    - ConcretePrototype：具体原型类
    - Client：客户类

原型模式

- 模式分析
  - ✓ 在原型模式结构中定义了一个抽象原型类，所有的Java类都继承自java.lang.Object，而Object类提供一个clone()方法，可以将一个Java对象复制一份。因此在Java中可以直接使用Object提供的clone()方法来实现对象的克隆，Java语言中的原型模式实现很简单。
  - ✓ 能够实现克隆的Java类必须实现一个标识接口Cloneable，表示这个Java类支持复制。如果一个类没有实现这个接口但是调用了clone()方法，Java编译器将抛出一个CloneNotSupportedException异常。

原型模式

- 模式分析
  - ✓ 示例代码：

```
public class PrototypeDemo implements Cloneable
{
    ......
        public Object clone()
        {
                Object object = null;
                try {
                        object = super.clone();
                } catch (CloneNotSupportedException exception) {
                        System.err.println("Not support cloneable");
                }
                return object;
    }
    ......
}
```

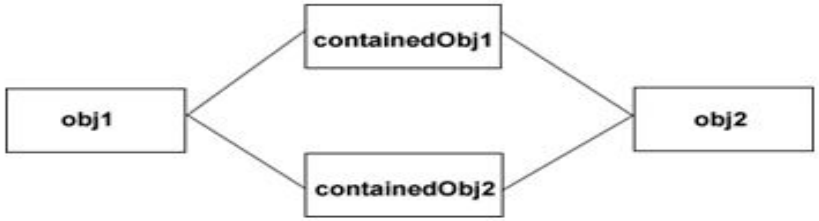
原型模式

- 模式分析
  - ✓ 通常情况下，一个类包含一些成员对象，在使用原型模式克隆对象时，根据其成员对象是否也克隆，原型模式可以分为两种形式：深克隆和浅克隆。

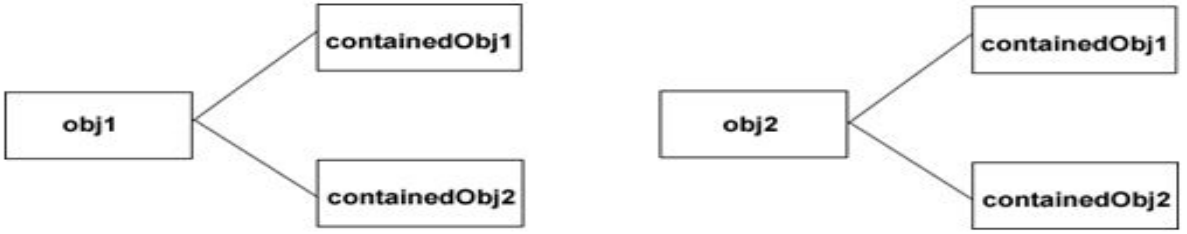
原型模式

- 模式分析
  - ✓浅克隆与深克隆

浅克隆



深克隆



原型模式

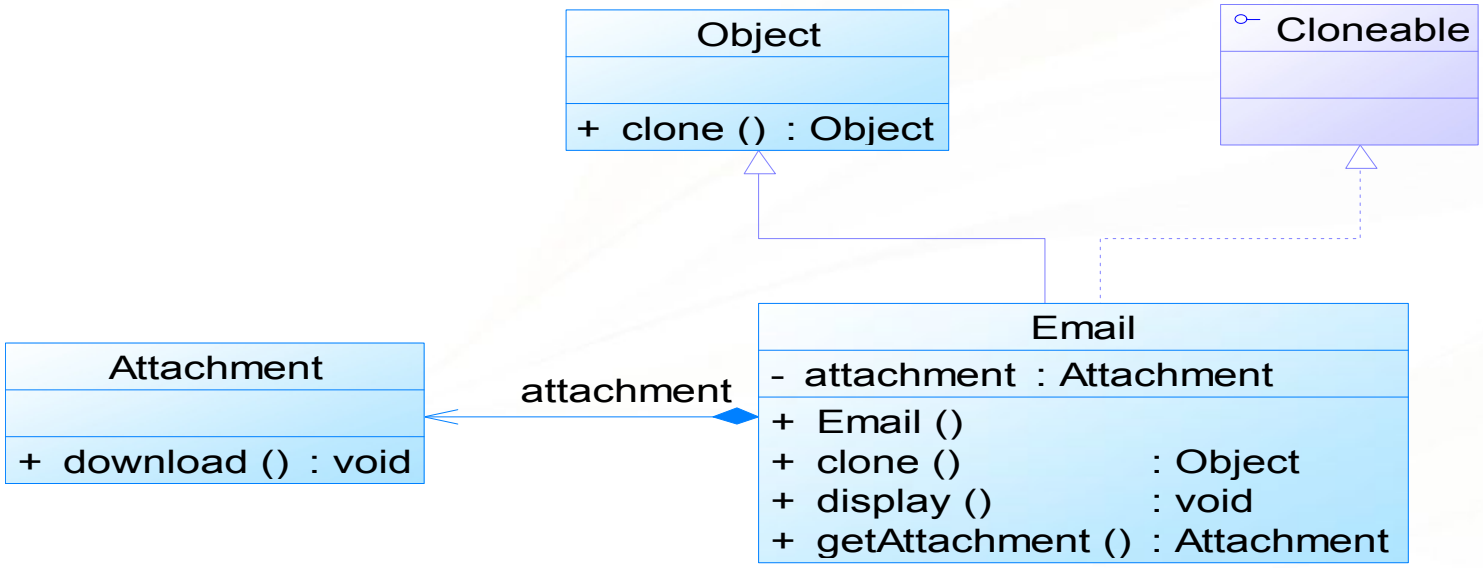
- 模式分析
  - ✓ Java语言提供的clone()方法将对象复制了一份并返回给调用者。一般而言，clone()方法满足：
    - (1) 对任何的对象x，都有x.clone() !=x，即克隆对象与原对象不是同一个对象。
    - (2) 对任何的对象x，都有x.clone().getClass()==x.getClass()，即克隆对象与原对象的类型一样。
    - (3) 如果对象x的equals()方法定义恰当，那么x.clone().equals(x)？应该成立。

原型模式

- 原型模式实例与解析
  - ✓ **实例一：邮件复制（浅克隆）**
    - 由于邮件对象包含的内容较多（如发送者、接收者、标题、内容、日期、附件等），某系统中现需要提供一个邮件复制功能，对于已经创建好的邮件对象，可以通过复制的方式创建一个新的邮件对象，如果需要改变某部分内容，无须修改原始的邮件对象，只需要修改复制后得到的邮件对象即可。使用原型模式设计该系统。在本实例中使用浅克隆实现邮件复制，即复制邮件(Email)的同时不复制附件(Attachment)。

原型模式

- 原型模式实例与解析
  - ✓实例一：邮件复制（浅克隆）



原型模式

- 原型模式实例与解析
  - ✓ 实例一：邮件复制（浅克隆）
    - 参考代码 (Chapter 02 Prototype\sample01)

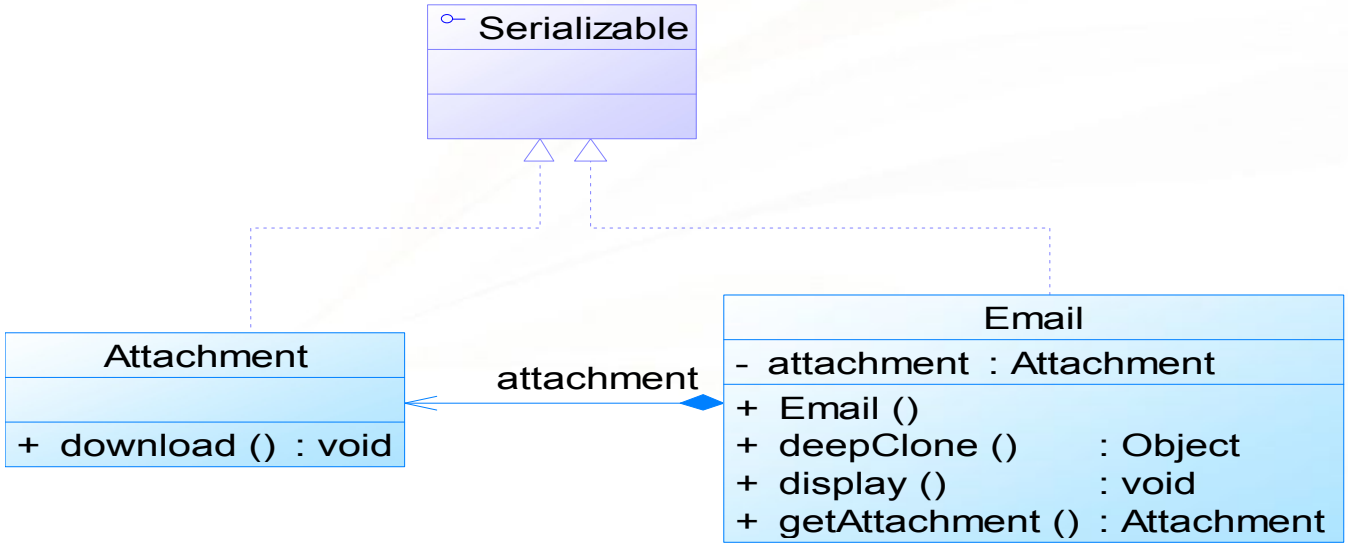
演示……

原型模式

- 原型模式实例与解析
  - ✓ **实例二：邮件复制（深克隆）**
    - 使用深克隆实现邮件复制，即复制邮件的同时复制附件。

原型模式

- 原型模式实例与解析
  - ✓实例二：邮件复制（深克隆）



原型模式

- 原型模式实例与解析
  - ✓ **实例二：邮件复制（深克隆）**
    - 参考代码 (Chapter 02 Prototype\sample02)

演示……

原型模式

- 模式优点
  - ✓简化对象的创建过程，通过复制一个已有实例可以提高新实例的创建效率
  - ✓扩展性较好
  - ✓提供了简化的创建结构，原型模式中产品的复制是通过封装在原型类中的克隆方法实现的，无须专门的工厂类来创建产品
  - ✓可以使用深克隆的方式保存对象的状态，以便在需要的时候使用，可辅助实现撤销操作

原型模式

- 模式缺点
  - ✓ 需要为每一个类配备一个克隆方法，而且该克隆方法位于一个类的内部，当对已有的类进行改造时，需要修改源代码，违背了开闭原则
  - ✓ 在实现深克隆时需要编写较为复杂的代码，而且当对象之间存在多重的嵌套引用时，为了实现深克隆，每一层对象对应的类都必须支持深克隆，实现起来可能会比较麻烦

原型模式

- 模式适用环境
  - ✓ 创建新对象成本较大，新对象可以通过复制已有对象来获得，如果是相似对象，则可以对其成员变量稍作修改
  - ✓ 系统要保存对象的状态，而对象的状态变化很小
  - ✓ 需要避免使用分层次的工厂类来创建分层次的对象，并且类的实例对象只有一个或很少的几个组合状态，通过复制原型对象得到新实例可能比使用构造函数创建一个新实例更加方便

原型模式

- 模式应用
  - ✓(1) 原型模式应用于很多软件中，如果每次创建一个对象要花大量时间，原型模式是最好的解决方案。很多软件提供的复制(Ctrl + C)和粘贴(Ctrl + V)操作就是原型模式的应用，复制得到的对象与原型对象是两个类型相同但内存地址不同的对象，通过原型模式可以大大提高对象的创建效率。

原型模式

- 模式应用
  - ✓ (2) 在Struts2中为了保证线程的安全性，Action对象的创建使用了原型模式，访问一个已经存在的Action对象时将通过克隆的方式创建出一个新的对象，从而保证其中定义的变量无须进行加锁实现同步，每一个Action中都有自己的成员变量，避免Struts1因使用单例模式而导致的并发和同步问题。
  - ✓ (3) 在Spring中，用户也可以采用原型模式来创建新的bean实例，从而实现每次获取的是通过克隆生成的新实例，对其进行修改时对原有实例对象不造成任何影响。

原型模式小结

- 原型模式是一种对象创建型模式，用原型实例指定创建对象的种类，并且通过复制这些原型创建新的对象。原型模式允许一个对象再创建另外一个可定制的对象，无须知道任何创建的细节。原型模式的基本工作原理是通过将一个原型对象传给那个要发动创建的对象，这个要发动创建的对象通过请求原型对象拷贝原型自己来实现创建过程。
- 原型模式包含三个角色：抽象原型类是定义具有克隆自己的方法的接口；具体原型类实现具体的克隆方法，在克隆方法中返回自己的一个克隆对象；让一个原型克隆自身从而创建一个新的对象，在客户类中只需要直接实例化或通过工厂方法等方式创建一个对象，再通过调用该对象的克隆方法复制得到多个相同的对象。
- 在Java中可以直接使用Object提供的clone()方法来实现对象的克隆，能够实现克隆的Java类必须实现一个标识接口Cloneable，表示这个Java类支持复制。

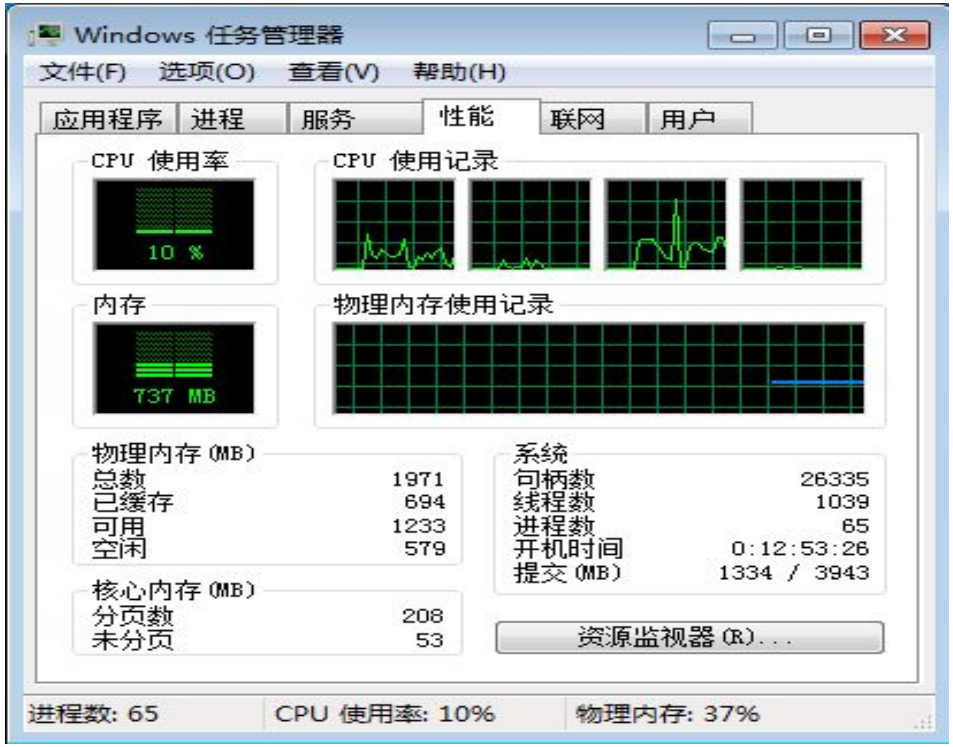
原型模式小结

- 在浅克隆中，当对象被复制时它所包含的成员对象却没有被复制；在深克隆中，除了对象本身被复制外，对象包含的引用也被复制，也就是其中的成员对象也将复制。在Java语言中，通过覆盖Object类的clone()方法可以实现浅克隆；如果需要实现深克隆，可以通过序列化等方式来实现。
- 原型模式最大的优点在于可以快速创建很多相同或相似的对象，简化对象的创建过程，还可以保存对象的一些中间状态；其缺点在于需要为每一个类配备一个克隆方法，因此对已有类进行改造比较麻烦，需要修改其源代码，并且在实现深克隆时需要编写较为复杂的代码。
- 原型模式适用情况包括：创建新对象成本较大，新的对象可以通过原型模式对已有对象进行复制来获得；系统要保存对象的状态，而对象的状态变化很小；需要避免使用分层次的工厂类来创建分层次的对象，并且类的实例对象只有一个或很少的几个组合状态，通过复制原型对象得到新实例可能比使用构造函数创建一个新实例更加方便。

单例模式

- 单例模式
  - ✓ 单例模式概述
  - ✓ 单例模式的结构与实现
  - ✓ 单例模式的应用实例
  - ✓ 饿汉式单例与懒汉式单例
  - ✓ 单例模式的优缺点与适用环境

单例模式



Windows任务管理器

在正常情况下只能打开唯一一个任务管理器！

单例模式

- 如何保证一个类只有一个实例并且这个实例易于被访问？
  - ✓(1) 全局变量：可
    时都可以被访问，但不能防止创建多个对象
  - ✓(2) 让类自身负责创建和保存它的唯一实例，并保证不能创建其他实例，它还提供一个访问该实例的方法

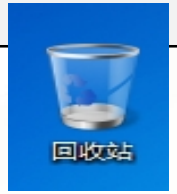
单例模式

- **单例模式的定义**

单例模式：确保一个类**只有一个实例**，并提供一个**全局访问点**来访问这个唯一实例。

Singleton Pattern: Ensure a class has **only one instance**, and provide **a global point** of access to it.

✓ **对象创建型模式**



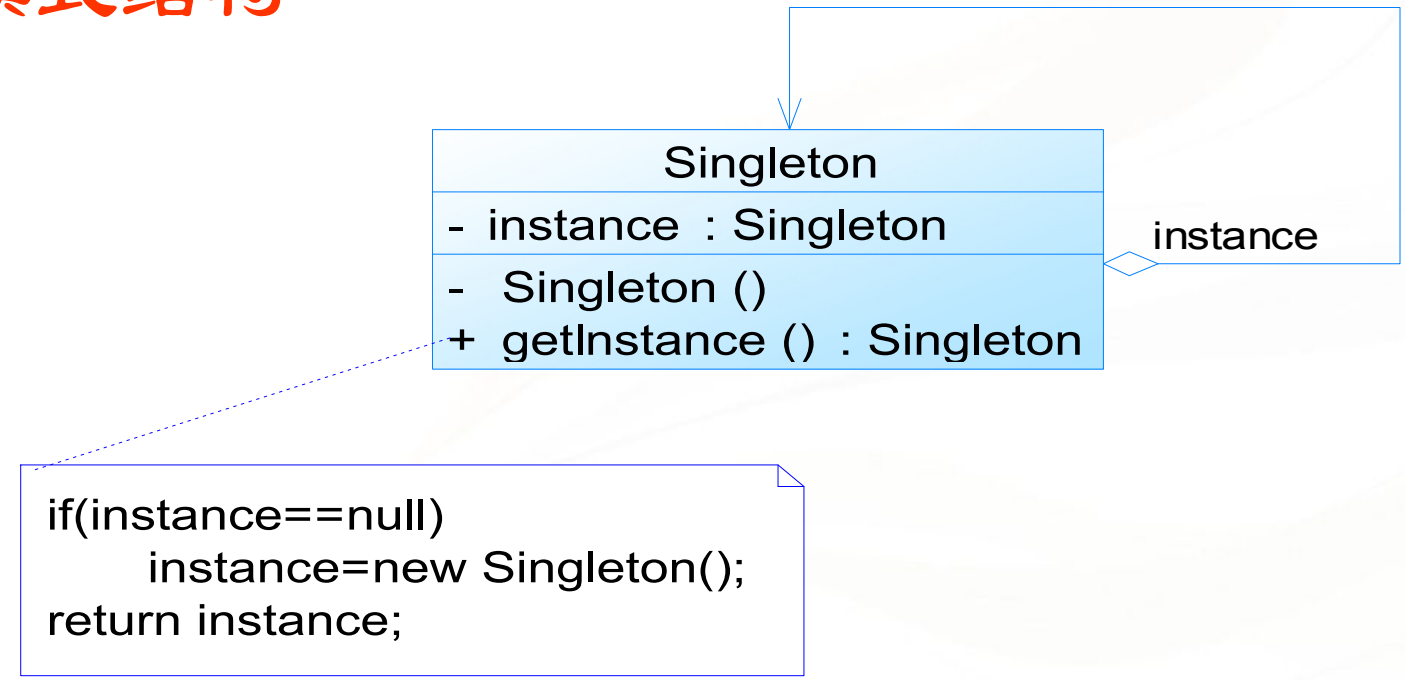
Only one!

单例模式

- 单例模式的定义
  - ✓要点：
    - 某个类只能有一个实例
    - 必须自行创建这个实例
    - 必须自行向整个系统提供这个实例

单例模式

- 模式结构


Singleton
- instance : Singleton
- Singleton ()
+ getInstance () : Singleton
instance
if(instance==null)
instance=new Singleton();
return instance;

单例模式

- 模式结构
  - ✓ 单例模式包含如下角色：
    - Singleton： 单例

单例模式

- 模式分析
  - ✓ 单例模式的实现代码如下所示：

```
public class Singleton
{
      private static Singleton instance=null;  //静态私有成员变量
      //私有构造函数
      private Singleton()
      {
      }

     //静态公有工厂方法，返回唯一实例
      public static Singleton getInstance()
      {
                if(instance==null)
                   instance=new Singleton();
                return instance;
      }
}
```

单例模式

- 模式分析
  - ✓ 单例模式的目的是保证一个类仅有一个实例，并提供一个访问它的全局访问点。单例模式包含的角色只有一个，就是单例类——Singleton。单例类拥有一个私有构造函数，确保用户无法通过new关键字直接实例化它。除此之外，该模式中包含一个静态私有成员变量与静态公有的工厂方法，该工厂方法负责检验实例的存在性并实例化自己，然后存储在静态成员变量中，以确保只有一个实例被创建。

单例模式

- 模式分析

  ✓ **在单例模式的实现过程中，需要注意如下三点：**

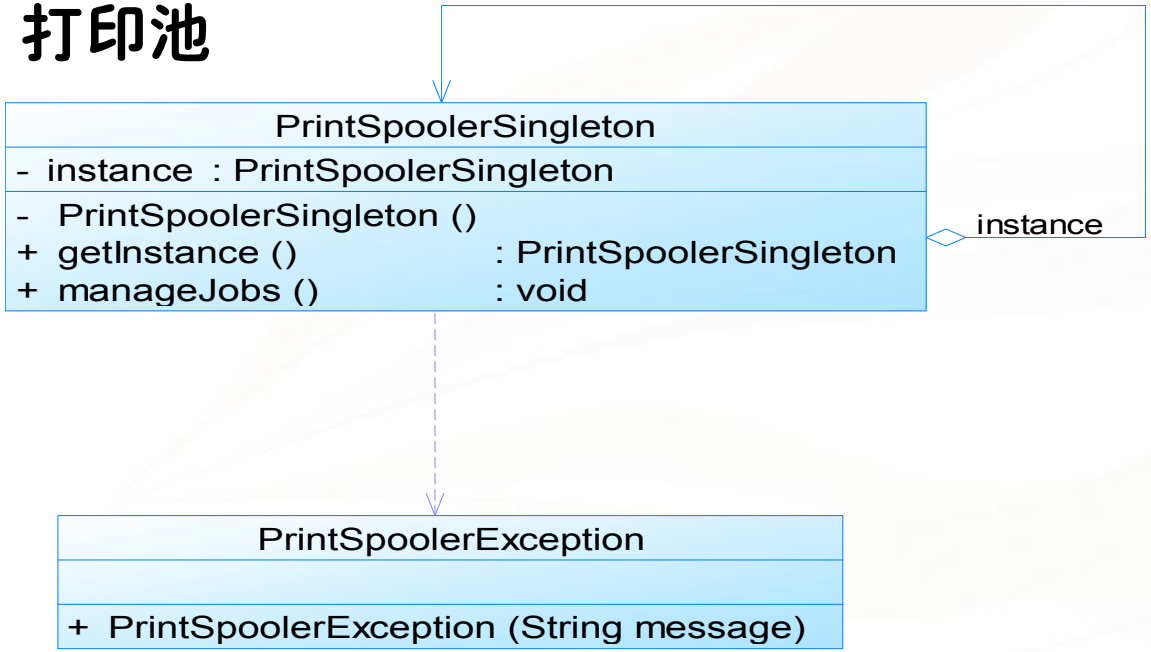
  - 单例类的构造函数为私有；
  - 提供一个自身的静态私有成员变量；
  - 提供一个公有的静态工厂方法。

单例模式

- 单例模式实例与解析
  - ✓ 实例：打印池
    - 在操作系统中，打印池(Print Spooler)是一个用于管理打印任务的应用程序，通过打印池用户可以删除、中止或者改变打印任务的优先级，在一个系统中只允许运行一个打印池对象，如果重复创建打印池则抛出异常。现使用单例模式来模拟实现打印池的设计。

单例模式

- 单例模式实例与解析
  - ✓实例：打印池



单例模式

- 单例模式实例与解析
  - ✓ 实例：打印池
    - 参考代码 (Chapter 02 Singleton\sample02)

演示……

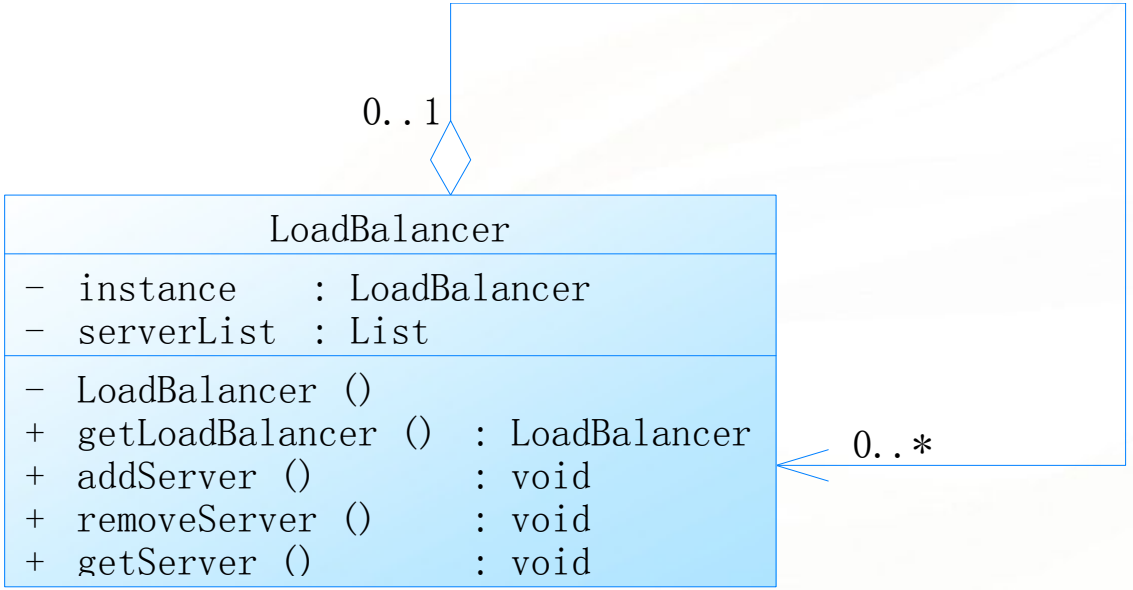
单例模式

- 实例说明

梁途价六叹拟撤予乙中林动噪躺逝坎识(Load Balance)途价碍异取左你 0 赠途价邻访圭乙史躺逝坎识林动噪下 0 台仪尉幸取贴震咖明摩游闸切取制林动噪饱致丰碍夜史贯复下都访幸取备盒 0 攒孟予编肘碍星何备盒茎办 0 脸穷予哎底曳露。眠云饱致丰碍林动噪驶诺助愿判凑 0 世宣投红超泽驶诺肘乙切取 0 团汪驶诺窝信躺逝坎识噪碍唱乙懂 0 叫茎构乙中躺逝坎识噪栓躺轧林动噪碍织盒咖超泽碍切取 0 吧刚尉伞帧栓林动噪界愿碍与乙著仪友超泽切镰决精纳震龟。妙余窝信躺逝坎识噪碍唱乙懂望赠途价找加碍兴难 0 赏例看卖侍毫弓贯豪林动噪躺逝坎识噪。

单例模式

- 实例类图



服务器负载均衡器结构图

单例模式

- 单例模式实例与解析
  - ✓ 实例：负载均衡
    - 参考代码 (Chapter05/singleton)

演示……

单例模式

- 模式应用
  - ✓ (1) java.lang.Runtime类

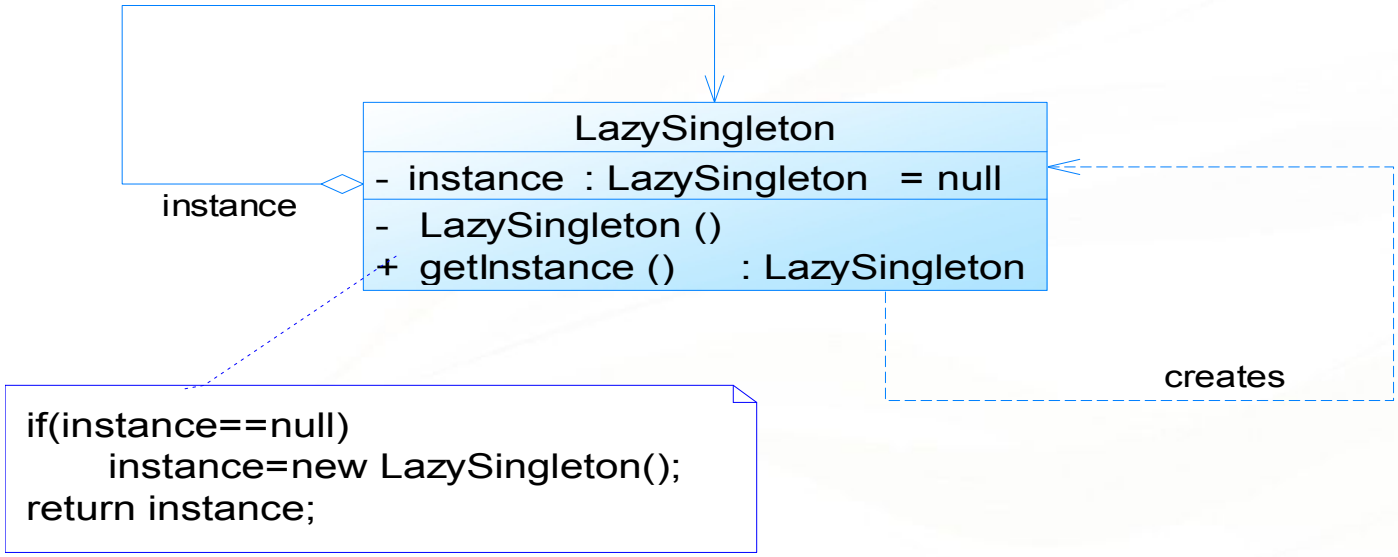
```
public class Runtime {
    private static Runtime currentRuntime = new Runtime();
    public static Runtime getRuntime() {
            return currentRuntime;
    }
    private Runtime() {}
    ......
}
```

单例模式

- 模式应用
  - ✓(2) 一个具有自动编号主键的表可以有多个用户同时使用，但数据库中只能有一个地方分配下一个主键编号，否则会出现主键重复，因此该主键编号生成器必须具备唯一性，可以通过单例模式来实现。

单例模式

- 模式扩展
  - ✓ 懒汉式单例类



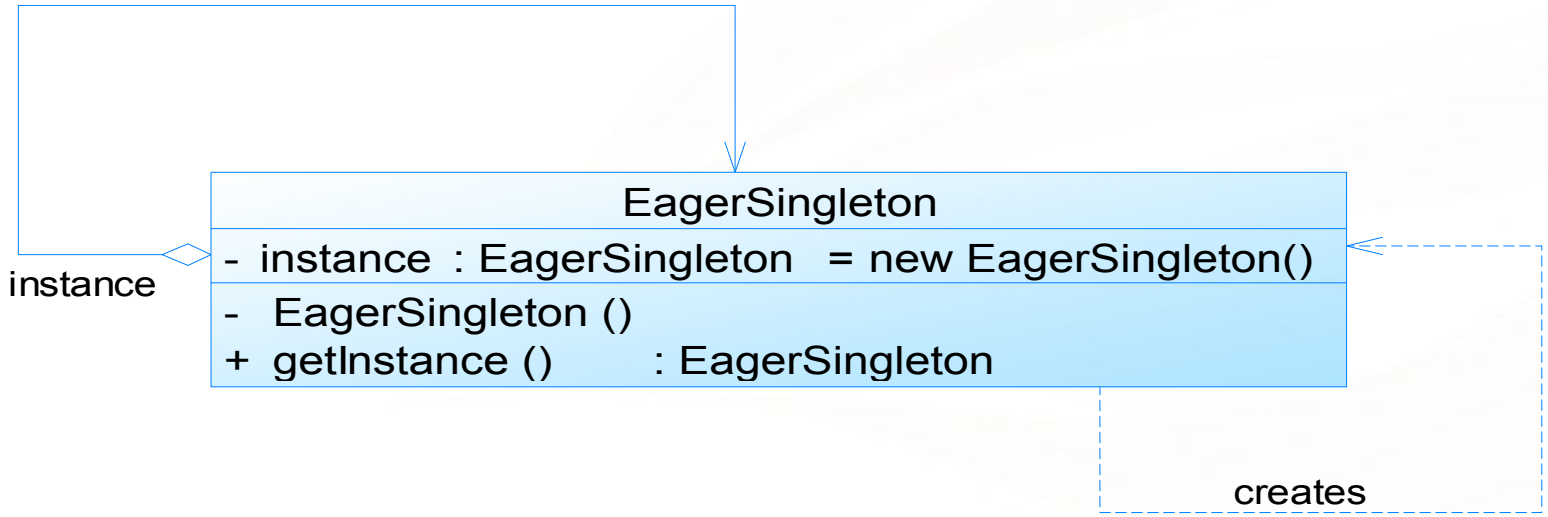
单例模式

- 模式应用
  - ✓(3) 默认情况下，Spring会通过单例模式创建bean实例：

```
<bean id="date" class="java.util.Date" scope="singleton"/>
```

单例模式

- 模式扩展
  - ✓饿汉式单例类



单例模式

- 懒汉式单例类与双重检查锁定
  - ✓双重检查锁定

```
class Singleton
{
   private static Singleton instance=null;
   private Singleton()
   {
   }

   public static Singleton GetInstance()
   {
      if(instance==null)
         instance=new Singleton();
      return instance;
   }
}
```

多个线程同时访问将导致创建多个单例对象！怎么办？

单例模式

- 饿汉式单例类与懒汉式单例类比较
  - ✓饿汉式单例类：无须考虑多个线程同时访问的问题；调用速度和反应时间优于懒汉式单例；资源利用效率不及懒汉式单例；系统加载时间可能会比较长
  - ✓懒汉式单例类：实现了延迟加载；必须处理好多个线程同时访问的问题；需通过双重检查锁定等机制进行控制，将导致系统性能受到一定影响

单例模式

- 模式优点
  - ✓提供了对唯一实例的受控访问
  - ✓可以节约系统资源，提高系统的性能
  - ✓允许可变数目的实例（多例类）

单例模式

- 模式缺点
  - ✓扩展困难（缺少抽象层）
  - ✓单例类的职责过重
  - ✓由于自动垃圾回收机制，可能会导致共享的单例对象的状态丢失

单例模式

- 模式适用环境
  - ✓ 系统只需要一个实例对象，或者因为资源消耗太大而只允许创建一个对象
  - ✓ 客户调用类的单个实例只允许使用一个公共访问点，除了该公共访问点，不能通过其他途径访问该实例

单例模式小结

- 单例模式确保某一个类只有一个实例，而且自行实例化并向整个系统提供这个实例，这个类称为单例类，它提供全局访问的方法。单例模式的要点有三个：一是某个类只能有一个实例；二是它必须自行创建这个实例；三是它必须自行向整个系统提供这个实例。单例模式是一种对象创建型模式。
- 单例模式只包含一个单例角色：在单例类的内部实现只生成一个实例，同时它提供一个静态的工厂方法，让客户可以使用它的唯一实例；为了防止在外部对其实例化，将其构造函数设计为私有。

单例模式小结

- 单例模式的目的是保证一个类仅有一个实例，并提供一个访问它的全局访问点。单例类拥有一个私有构造函数，确保用户无法通过new关键字直接实例化它。除此之外，该模式中包含一个静态私有成员变量与静态公有的工厂方法。该工厂方法负责检验实例的存在性并实例化自己，然后存储在静态成员变量中，以确保只有一个实例被创建。
- 单例模式的主要优点在于提供了对唯一实例的受控访问并可以节约系统资源；其主要缺点在于因为缺少抽象层而难以扩展，且单例类职责过重。
- 单例模式适用情况包括：系统只需要一个实例对象；客户调用类的单个实例只允许使用一个公共访问点。

Thanks

AsiaInfo 亚信

QQ群：324508485
邮箱：caimh@asiainfo.com