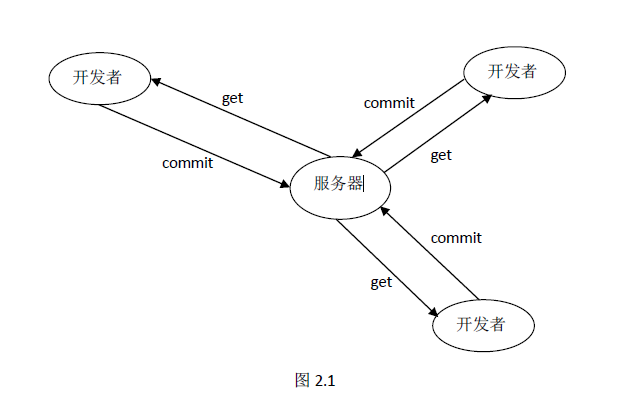
## Git相关知识：

git和svn 最大的差异在于**git是分布式的管理方式而svn是集中式的管理方式**。**分布式和集中式的最大区别在于开发者可以在本地提交。每个开发者机器上都有一个服务器的数据库。Git更加适合于多人开发。**

集中式管理的工作流程如下图（图2.1）：



集中式代码管理的核心是服务器，所有开发者在开始新一天的工作之前必须从服务器获取代码，然后开发，最后解决冲突，提交。所有的版本信息都放在服务器上。如果脱离了服务器，开发者基本上是不可以工作。

开始新一天的工作：

1：从服务器下载项目组最新代码。

2：进入自己的分支，进行工作，每隔1个小时向服务器自己的分支提交一次代码（很多人都有这个习惯。因为有时候自己对代码改来改去，最后又想还原到前一个小时的版本，或者看看前一个小时自己修改了哪些代码，就需要这样做了）。

3：下班时间快到了，把自己的分支合并到服务器主分支上，一天的工作完成，并反映给服务器。

这就是经典的svn工作流程，从流程上看，有不少缺点，但也有优点。

缺点：

1、服务器压力太大，数据库容量暴增。

2、如果不能连接到服务器上，基本上不可以工作，看上面第二步，如果服务器不能连接上，就不能提交，还原，对比等等。

3、不适合开源开发（开发人数非常非常多，但是Google app engine就是用svn的）。但是一般集中式管理的有非常明确的权限管理机制（例如分支访问限制），可以实现分层管理，从而很好的解决开发人数众多的问题。

优点：

1、管理方便，逻辑明确，符合一般人思维习惯。

2、易于管理，集中式服务器更能保证安全性。

3、代码一致性非常高。

4、适合开发人数不多的项目开发。

分布式管理的工作流程，如下图（图2.2）：

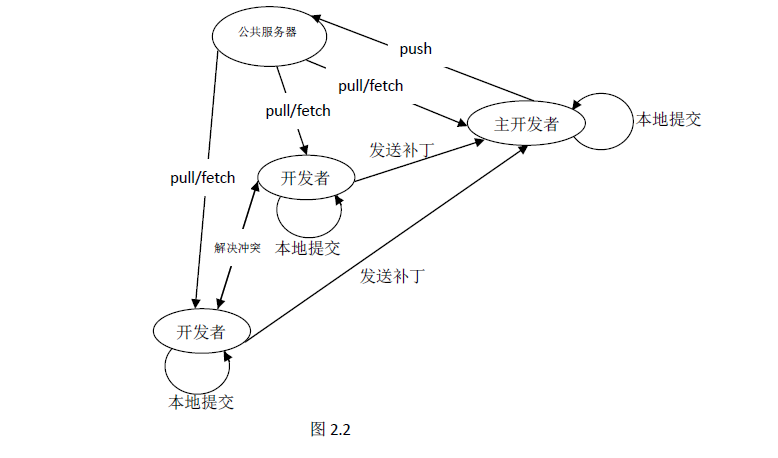


　图2.2就是经典的git开发过程。步骤如下：

**一般开发者的角度：**

1：从服务器上克隆数据库（包括代码和版本信息）到单机上。

2：在自己的机器上创建分支，修改代码。

3：在单机上自己创建的分支上提交代码。

4：在单机上合并分支。

5：新建一个分支，把服务器上最新版的代码fetch下来，然后跟自己的主分支合并。

6：生成补丁（patch），把补丁发送给主开发者。

7：看主开发者的反馈，如果主开发者发现两个一般开发者之间有冲突（他们之间可以合作解决的冲突），就会要求他们先解决冲突，然后再由其中一个人提交。如果主开发者可以自己解决，或者没有冲突，就通过。

8：一般开发者之间解决冲突的方法，开发者之间可以使用pull命令解决冲突，解决完冲突之后再向主开发者提交补丁。

**主开发者的角度（假设主开发者不用开发代码）：**

1：查看邮件或者通过其它方式查看一般开发者的提交状态。

2：打上补丁，解决冲突（可以自己解决，也可以要求开发者之间解决以后再重新提交，如果是开源项目，还要决定哪些补丁可用，哪些不用）。

3：向公共服务器提交结果，然后通知所有开发人员。

**优点：**

**适合分布式开发，强调个体。**

**公共服务器压力和数据量都不会太大。**

**速度快、灵活。**

**任意两个开发者之间可以很容易的解决冲突。**

**区别：**

1。速度：  
克隆一份全新的目录，以同样拥有五个（才五个）分支来说，SVN是同时复制5个版本的文件，也就是说重复五次同样的动作。而Git只是获取文件的每个版本的元素，然后只载入主要的分支（master）。在我的经验，克隆一个拥有将近一万个提交（commit），五个分支，每个分支有大约1500个文件的SVN，耗了将近一个小时！而Git只用了区区的1分钟！

2。版本库（repository）：  
据我所知，**SVN只能有一个指定中央版本库。当这个中央版本库有问题时，所有工作成员都一起瘫痪直到版本库维修完毕或者新的版本库设立完成**。

而**Git可以有无限个版本库。或者，更正确的说法，每一个Git都是一个版本库，区别是它们是否拥有活跃目录（Git Working Tree）。**如果主要版本库（例如：置于GitHub的版本库）发生了什么事，工作成员仍然可以在自己的本地版本库（local repository）提交，等待主要版本库恢复即可。工作成员也可以提交到其他的版本库！

3。分支（Branch）  
**在SVN，分支是一个完整的目录。且这个目录拥有完整的实际文件。**如果工作成员想要开启新的分支，那将会影响“全世界”！每个人都会拥有和你一样的分支。如果你的分支是用来进行破坏工作（安检测试），那将会像传染病一样。

而**Git，每个工作成员可以任意在自己的本地版本库开启无限个分支。**举例：当我想尝试破坏自己的程序（安检测试），并且想保留这些被修改的文件供日后使用，我可以开一个分支，做我喜欢的事。完全不需担心妨碍其他工作成员。只要我不合并及提交到主要版本库，没有一个工作成员会被影响。等到我不需要这个分支时，我只要把它从我的本地版本库删除即可。无痛无痒。  
Git的分支名是可以使用不同名字的。例如：我的本地分支名为testing，而在主要版本库的名字其实是master。  
最值得一提，我可以在**Git的任意一个提交点（commit point）开启分支**！（其中一个方法是使用gitk –all可观察整个提交记录，然后在任意点开启分支。）

4。提交（Commit）  
**在SVN，当你提交你的完成品时，它将直接记录到中央版本库。**当你发现你的完成品存在严重问题时，你已经无法阻止事情的发生了。如果网路中断，你根本没办法提交！

而**Git的提交完全属于本地版本库的活动。而你只需“推”（git push）到主要版本库即可。Git的“推”其实是在执行“同步”（Sync）。**

5。重新设立起点（Rebase）  
我没在SVN尝试过，不知道有没有这样的功能。

在Git，如果你想把别人的最新提交设立为现在这个分支的起点，只要执行git rebase branch\_name即可。这个和合并（merge）不同点是，merge会依据修改的时间视为最新，而Rebase会要求你去解决双方都有修改过的地方的矛盾（conflict）。A - B - E \- C - D A - B - E \ - C - D

6。系统档案  
**SVN会在每一个目录置放一个.svn。**如果想移除这些.svn是很累的。  
而**Git会在目录起点拥有一个.git目录**，以及.gitignore。

## Spring相关知识

**------开源的，控制反转（Inversion of Control）和面向切面的容器框架。**

spring是用来管理你的类的框架，管理它们之间的互相调用和依赖关系，通过接口解藕，可以方便的更换使用多种不同的技术实现来完成你的系统。

**Spring核心：控制反转+面向切面**

**IOC（控制反转）**-------创建对象并且组装对象之间的关系。

IOC:应用程序本身不负责依赖对象的创建和维护，**依赖对象的创建和维护是由外部容器负责的。**这样控制权就由应用程序转移到了外部容器。**控制权的转移就是所谓的反转。对象之间的依赖关系在后期通过配置文件（典型为 XML 文件）生成， Spring 里实现了两种注入方式：构造函数注入、 Setter 方法注入。**依赖注入(DI机制)：Dependency Injection：在运行时由外部容器将依赖对象注入到组件中。

**什么是依赖注入，依赖注入的作用是什么?**   
IOC是一种思想，它能指导我们如何设计出松耦合、更优良的程序。传统应用程序都是由我们在类内部主动创建依赖对象，从而导致类与类之间高耦合，难于测试；有了IOC容器后，把创建和查找依赖对象的控制权交给了容器，由容器进行注入组合对象，所以对象与对象之间是松散耦合，这样也方便测试，利于功能复用，更重要的是使得程序的整个体系结构变得非常灵活。

依赖注入的作用：减少类间耦合度，避免用new来创建对象

**Spring MVC工作原理**  
1. 客户端请求提交到**DispatcherServlet**  
2. 由DispatcherServlet控制器查询一个或多个HandlerMapping，找到处理请求的**Controller**  
3. DispatcherServlet将请求提交到Controller  
4. Controller调用业务逻辑处理后，返回**ModelAndView**  
5. DispatcherServlet查询一个或多个**ViewResoler视图解析器**，找到ModelAndView指定的视图  
6. 视图负责将结果显示到客户端

Spring Web MVC优势

1、清晰的角色划分：前端控制器（DispatcherServlet）、 请求到处理器映射（HandlerMapping）、处理器适配器（HandlerAdapter）、视图解析器（ViewResolver）、处理器或 页面控制器（Controller）、验证器（   Validator）、命令对象（Command  请求参数绑定到的对象就叫命令对象）、表单对象（Form Object 提供给表单展示和提交到的对象就叫表单对象）。

2、分工明确，而且扩展点相当灵活，可以很容易扩展，虽然几乎不需要；

3、由于命令对象就是一个POJO，无需继承框架特定API，可以使用命令对象直接作为业务对象；

4、和Spring 其他框架无缝集成，是其它Web框架所不具备的；

5、可适配，通过HandlerAdapter可以支持任意的类作为处理器；

6、可定制性，HandlerMapping、ViewResolver等能够非常简单的定制；

7、功能强大的数据验证、格式化、绑定机制；

8、利用Spring提供的Mock对象能够非常简单的进行Web层单元测试；

9、本地化、主题的解析的支持，使我们更容易进行国际化和主题的切换。

10、强大的JSP标签库，使JSP编写更容易。

………………还有比如RESTful风格的支持、简单的文件上传、约定大于配置的契约式编程支持、基于注解的零配置支持等等。

**为什么使用Spring**

1：**降低组件之间的耦合度，实现各层之间的解耦**

2：可以使用容器提供的众多服务。如：

--事务管理服务

3：提供了**单例模式**支持。开发人员不需要自己编写实现代码

4：提供了AOP技术。实现如：权限拦截，运行监控等功能

5：提供了众多的辅助类。如JDBC Template，HIbernate Template

6：对主流的应用框架提供了集成支持。集成Struts，JPA，Hibernate

**为什么要用Spring**   
1、Spring能很好的与各大框架进行集成  
2、创建对象时，如果我们不用spring需要用工厂模式来创建，这个spring相当于工厂模式已经帮我们做了创建对象的功能(IOC、依赖注入)。  
3、在用Hibernate的时候，如果不用spring每次都要写事务的提交代码，有了spring可以通过AOP帮助我们管理事务   
4、面向切面编程（AOP）在要记录日志的时候添加一条记录后需要在数据里同时添加一条添加成功了或失败的记录，那么就可以用Spring的Aop来处理，虽然不用Aop也能做但是不用Spring的Aop就会写很多重复的代码。

**spring 的优点？**   
1.降低了组件之间的耦合性 ，实现了软件**各层之间的解耦**  
2.可以使用容易提供的**众多服务**，如事务管理，消息服务等   
3.容器提供**单例模式**支持   
4.容器提供了**AOP技术**，利用它很容易实现如权限拦截，运行期监控等功能   
5.容器提供了众多的辅助类，能加快应用的开发   
6.spring对于**主流的应用框架提供了集成支持**，如hibernate，JPA，Struts等   
7.spring属于低侵入式设计，代码的污染极低   
8.独立于各种应用服务器   
9.spring的DI机制降低了业务对象替换的复杂性   
10.Spring的**高度开放性**，并不强制应用完全依赖于Spring，开发者可以自由选择spring的部分或全部

**spring实现的2种设计模式**

**单态模式**：接受容器管理的bean，默认采用单态模式管理。

**工厂模式**：spring就是实例化和管理全部bean的工厂。spring的上下文也就是工厂，也被设计成单态的

**spring容器：BeanFactory和ApplicationContext**之间是父子的关系

**Spring实例化bean的三种方式**  
1：默认使用类加载器   
2：使用静态工厂方法实例化bean对象   
<bean id="名称" class="工厂类" factory-method="工厂类中创建bean实例的方法"/>   
3：使用实例工厂方法实例化bena对象   
<bean id="实例工厂名称" class="工厂类的路径"/>   
<bean id="名称"  factory-bean="实例工厂名称" method="实例工厂里边的创建bena对象的方法"/>

**Spring管理的bean的作用域**Spring生成的bean默认为单实例。singleton。   
如果需要每次调用Spring的getBean方法就创建一个 bean的实例，可指定bean的属性scope="prototype"。还有：request、session、global session 这3个一般基于web的spring ApplicationContext情形下有效。

**四、介绍一下Spring的事务管理**   
事务就是对一系列的数据库操作（比如插入多条数据）进行统一的提交或回滚操作，如果插入成功，那么一起成功，如果中间有一条出现异常，那么回滚之前的所有操作。这样可以防止出现脏数据，防止数据库数据出现问题。

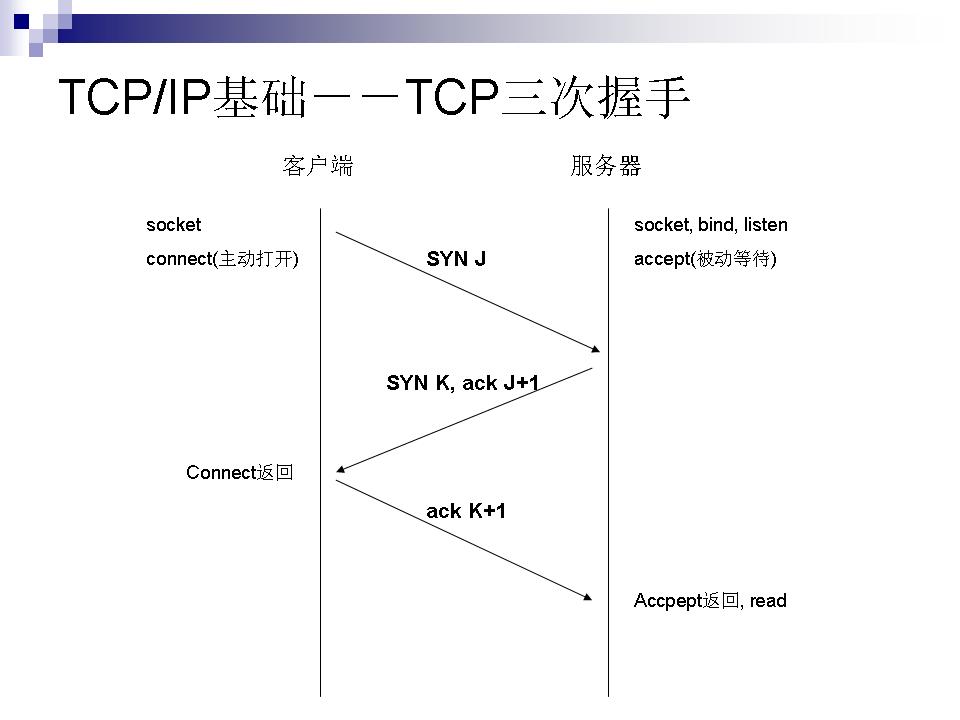
开发中为了避免这种情况一般都会进行事务管理。**Spring中也有自己的事务管理机制，一般是使用TransactionMananger进行管理，可以通过Spring的注入来完成此功能。**Spring配置文件中关于事务配置总是由三个组成部分，分别是**DataSource**、**TransactionManager**和**代理机制**这三部分，无论哪种配置方式，一般变化的只是代理机制这部分。DataSource、TransactionManager这两部分只是会根据数据访问方式有所变化，比如使用Hibernate进行数据访问时，DataSource实际为SessionFactory，TransactionManager的实现为HibernateTransactionManager。

使得我们再也无需要去处理获得连接、关闭连接、事务提交和回滚等这些操作。再也无需要我们在与事务相关的方法中处理大量的try…catch…finally代码。

**Spring的事务管理器** Spring的事务管理器有5个，都实现了PlatformTransactionManager接口，如下所示：    
DataSourceTransactionManager          **JDBC事务管理器**   
HibernateTransactionManager            **Hibernate事务管理器**JdoTransactionManager                  **JDO事务管理器**   
JtaTransactionManager                   JTA事务管理器        
PersistenceBrokerTransactionManager    Apache的OJB事务管理器

**Spring中的BeanFactory与ApplicationContext的作用有哪些?**  
1、BeanFactory负责读取bean的配置文件，管理bean的加载、实例化，维护bean之间的依赖关系，负责bean的生命周期。   
2、**ApplicationContext**除了提供上述BeanFactory所能提供的功能之外，还提供了更完整的框架功能： a. 国际化支持b. 资源访问 c. 事件传递

**hibernate整体流程   
1：通过configuration来读cfg.xml文件   
2：得到SessionFactory 工厂   
3：通过SessionFactory 工厂来创建Session实例   
4：通过Session打开事务   
5：通过session的api操作数据库   
6：事务提交   
7：关闭连接   
  
为什么要用：   
1. 对JDBC访问数据库的代码做了封装，大大简化了数据访问层繁琐的重复性代码。     
2. Hibernate是一个基于JDBC的主流持久化框架，是一个优秀的ORM实现。他很大程度的简化DAO层的编码工作。     
3. hibernate使用Java反射机制，而不是字节码增强程序来实现透明性。     
4. hibernate的性能非常好，因为它是个轻量级框架。映射的灵活性很出色。它支持各种关系数据库，从一对一到多对多的各种复杂关系。**



在TCP/IP协议中，TCP协议提供可靠的连接服务，采用三次握手建立一个连接。   
第一次握手：建立连接时，客户端发送syn包(syn=j)到服务器，并进入SYN\_SEND状态，等待服务器确认；   
第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个SYN包（syn=k），即SYN+ACK包，此时服务器进 入SYN\_RECV状态；

第三次握手：客户端收到服务器的SYN＋ACK包，向服务器发送确认包ACK(ack=k+1)，此包发送完毕，客户端和服务器进入 ESTABLISHED状态，完成三次握手。

完成三次握手，客户端与服务器开始传送数据. 握 手过程中传送的包里不包含数据，三次握手完毕后，客户端与服务器才正式开始传送数据。理想状态下，TCP连接一旦建立，在通信双方中的任何一方主 动关闭连接之前，TCP 连接都将被一直保持下去。断开连接时服务器和客户端均可以主动发起断开TCP连接的请求，断开过程需要经过“四次握手”（过程就不细写了，就是服务器和客户端交互，最终确定断开）

**各种排序的稳定性，时间复杂度和空间复杂度总结：**

****

希尔排序又叫**缩小增量排序**

**基本思想：**

先将整个待排序的记录序列分割成为若干子序列分别进行直接插入排序，待整个序列中的记录“基本有序”时，再对全体记录进行依次直接插入排序。

