       今天晚上2012年欧洲杯决赛（西班牙 VS 意大利），作为一名铁杆球迷，偶当然不会错过（请不要让我来预测比分，我不是章鱼，更何况这两个队我都非常喜欢，输赢我都很淡定，微笑），在静候决赛的这段时间，突然萌发一个想法，将设计模式跟足球联系到一起写点啥，就像当年那篇知名度极高的《追MM与设计模式》一样，以供娱乐！吐舌头，话不多说，即刻动手！

**创建型模式**

**(1) 工厂方法模式**：近年来大型足球比赛（世界杯和欧洲杯）的指定用球都是阿迪达斯的（据说是签了合同的），当然Adidas足球是由Adidas公司生产的，除此之外，Nike公司也生产Nike足球，KAPPA（背靠背）公司也生产背靠背足球，足球生产商是工厂，足球是产品。增加一种新的足球品牌，对应需要增加一个新的生产商。

      工厂方法模式(Factory Method)：定义一个用于创建对象的接口，让子类决定将哪一个类实例化。工厂方法模式使一个类的实例化延迟到其子类。

**(2) 抽象工厂模式**：Adidas工厂除了生产Adidas足球外，还生产Adidas球鞋、球服、球袜（adidas is all in）；Nike工厂也生产Nike足球、球鞋、球服、球袜等，在此，Adidas和Nike是工厂，同一品牌的足球、球鞋、球服、球袜构成了一个产品族，一个工厂可以生产一族产品，而不只是一种产品。

       抽象工厂模式(Abstract Factory): 提供一个创建一系列相关或相互依赖对象的接口，而无须指定它们具体的类。

**(3) 单例模式**：在比赛过程中（在场上的，替补不算）每个球队的守门员有且仅有一个，肯定不会有两个穿相同球衣的守门员同时上场，这不是单例吗？如果布冯或者卡西能出场，还有哪个意大利或者西班牙守门员敢去跟他们抢首发呢？？布冯你是唯一的！卡西，你也是！当然，皮尔洛也是，哈维也是，小法也是，巴神也是......原来有这么多单例。

      单例模式(Singleton): 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。

**(4) 建造者模式**：如果我要一套完整的意大利国家队足球装备（除球鞋，这个貌似没有统一）：包括球衣、球裤、球袜，只需跟某专卖店销售人员说一下（想象，想象......）：我要一套意大利队的足球装备，大小为XL，返回给你的是一套经典的蓝色意大利国家队队服，包括蓝色的足球袜；当然你的朋友可以说他要一套西班牙队的足球装备，返回给他的是一套红色的斗牛士足球装备，袜子，当然也是红的。在此，销售人员相当于建造者模式中的指挥者(Director)，他向用户返回一个复杂产品(足球装备)，该复杂产品由多个部件组成(球衣、球裤、球袜等)，用户无须关心具体组装过程即可得到一个完整的复杂产品。

       建造者模式(Builder): 将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。

**(5) 原型模式**：无论是足球还是球服，都是批量生产的，例如2012年欧洲杯的指定用球Adidas 探戈12（Tango 12），先做一个原型（模板），然后照着生产就好了，想要多少就生产多少。

      原型模式(Prototype):用原型实例指定创建对象的种类，并且通过拷贝这个原型来创建新的对象。

**结构型模式**

**(6) 适配器模式**：很多足球队都喜欢请外国教练（其中有一支我们都非常熟悉的国家队，名字偶就不说了，大家都懂的，微笑），外国教练请回来通常很难跟队员直接交流（语言不通），因此需要配翻译，此时，翻译充当了教练和队员之间的适配器，负责协调教练和队员之间的交流。

       例如：pass --> shoot --> goal **转换**  传球 --> 射门 --> 进球

      适配器模式(Adapter): 将一个类的接口转换成用户希望的另一个接口，使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。

**(7) 桥接模式**：在足球比赛中，有人踢前锋、有人踢中场（前腰、中卫）、有人踢后卫；当然，有人习惯踢左边、有人习惯踢右边、也有人喜欢站在中间，因此诞生了左中卫、右前锋、中后卫、右后卫等名词，难道这不是两个变化维度的组合吗？

       桥接模式(Bridge): 将抽象部分与实现部分分离，使它们都可以独立地变化。

**(8) 组合模式**：2012年欧洲杯一共分为四个组，每个组四个队，每个队有23名球员，如果要用一个图来表示2012年欧洲杯全体球员及各国分组情况，不用说，一定是个树状图，组里有队，队里有人，如果想要召开B组（赛前公认的死亡之组）队员大会，在B组的节点上写下通知：“下午3点，召开重要会议，事关出线！”，想必荷兰、德国、葡萄牙、丹麦队员都会积极响应，随叫这几个“苦逼”队位于同一个节点的分支上呢？微笑

     组合模式(Composite): 将对象组合成树形结构以表示“部分-整体”的层次结构，它使得客户对单个对象和复合对象的使用具有一致性。

**(9) 装饰模式**：现在足球服上的广告越来越多了，2012年欧洲杯夺冠热门之一（赛前预测）德国队队服胸前右边一个奔驰，左边一个阿迪，当然还可以继续增加，广告既没有改变球衣的用途和性能，还能起到装饰效果，增加收入，何乐而不为呢？就是半决赛没能够让巴神继续“思考人生”，悲催的德国队！增加新的广告，只需对原有球服继续装饰即可。

      装饰模式(Decorator): 动态地给一个对象添加一些额外的职责，就扩展功能而言， 它比生成子类的方式更为灵活。

**(10) 外观模式**：为了给记者和球队（球员、教练等）提供一个交流的平台，欧洲杯组委会在每场足球比赛前后都安排了新闻发布会，记者可以通过新闻发布会来与球队进行沟通交流（虽然不是每个队员会出现在新闻发布会上），在此，新闻发布会充当了记者（客户端）和队员、教练（子系统）之间的外观角色。当然，新闻发布会并不会影响某位记者单独采访某位球员（这一点也与外观类的定义一致，微笑）。

      外观模式(Facade): 子系统中的一组接口提供一个一致的界面，定义一个高层接口，这个接口使得这一子系统更加容易使用。

**(11) 享元模式**：同一个国家队的队员，他们都共享着一个伟大的称谓，即"XXX国家队队员"，例如“意大利国家队队员”、“西班牙国家队队员”（一说到“中国国家队队员”就伤心，还是不说了，难过），因此，"XXX国家队队员"是一个可以共享的内部状态。但是在比赛过程中，每个队员身披不同号码的球衣，球衣号码是不能共享的，同一个国家队的队员每个人都拥有不同的号码，因此，球衣号码是不能够共享的外部状态。在享元模式中区分了对象的内部状态和外部状态。

      享元模式(Flyweight): 运用共享技术有效地支持大量细粒度的对象。

**(12) 代理模式**：足球场外，球员转会是一个热门话题。转会当然离不开球员的经纪人，经纪人将球员的想法传递给另一家俱乐部。经纪人就是球员的代理，球员是目标对象，而经纪人是代理对象，经纪人隔离了球员和“客户端”，“拍广告，请找我的经纪人”，“采访，请找我的经纪人”......

      代理模式(Proxy)：为其他对象提供一个代理以控制对这个对象的访问。

**行为型模式（上）**

**(13) 职责链模式**：布冯手抛球给基耶利尼、基耶利尼传给皮尔洛、皮尔洛带球过人之后将球直塞给快速插上的巴洛特利，巴洛特利倒钩射门，球进了，球进了，又是巴洛特利，巴洛特利立功了，伟大的意大利前锋！他继承了意大利的光荣传统，巴乔、因扎吉、皮耶罗在这一刻灵魂附体！巴洛特利代表了意大利足球悠久的历史和传统，在这一刻他不是一个人在战斗，他不是一个人！大笑

      在此，足球就是一个请求，而球员就是请求的处理者，足球在球员间不断进行传递，构成了一条传递链。

      职责链模式(Chain of Responsibility): 为解除请求的发送者和接收者之间耦合，而使多个对象都有机会处理这个请求；将这些对象连成一条链，并沿着这条链传递该请求，直到有一个对象处理它。

**(14) 命令模式**：普兰德利掌心朝外，向前一推，意大利全线压上；普兰德利掌心朝内，向后一拨，意大利全线退防。作为意大利主教练，普兰德利就是命令的发送者，手势就是命令对象，所有队员都是命令的接收者。不同的命令对象将对应不同的执行动作。

      命令模式(Command): 将一个请求封装为一个对象，从而可用不同的请求对客户进行参数化；对请求排队或记录请求日志，以及支持可取消的操作。

**(15) 解释器模式**：在足球场上，教练的手势就是一门语言，有的表示“传球”，有的表示“全线压上”，有的表示“全线防守”，每个队员都需要在比赛中阅读教练的手势并将其转换成执行指令，按照教练的意图来展开攻守。

      解释器模式(Interpreter): 定义语言的文法，并且建立一个解释器来解释该语言中的句子。

**(16) 迭代器模式**：下面出场的是西班牙队：1号守门员卡西利亚斯、3号后卫皮克、6号中场球员伊内斯塔、8号哈维、9号托雷斯......一个个来，不急，这次是按照球衣号码，下次再按照位置从前到后、从左到右介绍一次。西班牙队是一个包含多个队员的聚合对象，可以提供一个迭代器来遍历其中的队员。

      迭代器模式(Iterator): 提供一种方法顺序访问一个聚合对象中各个元素，而又不需暴露该对象的内部表示。

**(17) 中介者模式**：比赛摔倒怎么办？看裁判；没顶到球怎么办？看裁判；被踢中要害部位怎么办？看裁判；球到底进没进，看裁判！。裁判经常是足球赛场的主角，当两队队员发生冲突时，裁判还是很重要滴，他充当了球员之间的中介者（调停者）。一切需服从裁判，他才是球场的老大！

      中介者模式(Mediator): 用一个中介对象来封装一系列的对象交互；中介者使各对象不需要显式地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互。

**行为型模式（下）**

**意大利昨晚太杯具了！！！不说了，继续把最后一部分写完。**

**(18) 备忘录模式：**足球是圆的，一切皆有可能发生。要是有后悔药的话，如果能回到昨晚2012年欧洲杯决赛的中场休息，我相信普兰德利一定不会用莫塔换下蒙托利沃；如果能回到昨晚比赛开始，我相信一开始就不会让基耶利尼上场，如果能回到......（再回可能意大利就被德国淘汰了，微笑）能回到吗？回不到哦，要是能回到过去的话我还真想再过一次20岁（回忆那段青葱岁月），可惜人生没有后悔药啊。幸好软件系统中可以通过备忘录模式来实现对象的状态恢复，备忘录就是软件中的后悔药，它就是软件中的月光宝盒。Ctrl + Z，撤销随你！

      备忘录模式(Memento): 在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态，这样以后就可将该对象恢复到先前保存的状态。

**(19) 观察者模式：**教练大手一挥，全线压上。此时，教练是观察目标，球员是观察者，观察目标与观察者之间有一对多的联动，当然裁判也可以看成是球员的观察目标，终场哨一吹，西班牙乐成一片，意大利哭成一片，不同的观察者反应还真的有所不同，大笑。

      观察者模式(Observer): 定义对象间的一种一对多的依赖关系，以便当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动更新。

**(20) 状态模式：**踢足球是要看状态的，当然状态是会转换的。有的球员上场时状态不行，老是“思考人生”，此时处于“梦游状态”；踢着踢着状态好起来了，头顶脚踢，运气好的话还能进球，此时处于“亢奋状态”；然后随着体力下降，动作变形，射门软绵无力，一碰就倒，此时又处于“体力透支状态”。随着比赛的进行，这几个状态会发生转换，而且在不同状态下球员的行为也不同，在梦游状态下基本上没有射门，在亢奋状态下基本上没有传球（全自己射了），在体力透支状态下基本上没有抢断（自己都拿不稳了，还抢啥抢）。如果将每一种状态封装在一个状态类中，那么球员就是拥有状态的环境类了。

       状态模式(State): 允许一个对象在其内部状态改变时改变它的行为，对象看起来似乎修改了它所属的类。

**(21) 策略模式：**据说1863年足球刚开始的时候流行1-0-9阵型，1个后卫，9个前锋，木有越位，惊讶。随着足球的发展，现代足球阵型的变化越来越多，面对防守型球队，可以选择3-5-3阵型，面对攻击性强的球队，可以选择5-4-1阵型，当然还有经典的4-4-2。每一种阵型都是一种策略，面对不同的对手可以选择不同的策略。

      策略模式(Strategy): 定义一系列的算法,把它们一个个封装起来，并且使它们可相互替换，策略模式使得算法的变化可独立于使用它的客户。

**(22) 模板方法模式：**“角球！这是意大利的机会，今天意大利面对全面占优的西班牙办法不多，定位球可能是最有效的破门方式了。皮尔洛开出一个战术角球，传给卡萨诺，卡萨诺传前点，马尔基西奥头球抢点，球顶高了。不过这次角球配合设计得很精妙，给西班牙带来了威胁，可惜整场比赛这种机会不多啊！！”。在战术角球中，A开球、B传球、C抢点再射门，这是一个战术的框架，当然C到底是抢前点还是抢后点可以根据实际情况来选择，如果将踢战术角球设计为一个模板方法，那么每一个步骤就是其中要调用的基本方法了，而且在不同战术中某些具体步骤的实施还各不相同。Perfect！真是模板方法模式的非典型应用！

      模板方法模式(Template Method): 定义一个操作中的算法的骨架，而将一些步骤延迟到子类中，使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。

**(23) 访问者模式：**有些从事足球比赛研究的人很喜欢数据。有的专门研究一场比赛中每个球员的跑动距离，有的研究每个球员的抢断次数，有的研究每个球员的射门次数，有的研究球员传球次数，有的研究球员传球成功率......如果将每一种研究类型看成一个访问者，那么球队就是一个包含多个队员元素的对象结构，以供不同访问者来研究，微笑，当然，我们还可以很方便地增加新的访问者（研究者），例如，研究每个球员在比赛中吐口水的次数，研究每个球员在比赛中与对方球员“亲密接触”次数......

      访问者模式(Visitor): 表示一个作用于某对象结构中的各元素的操作，可以在不改变各元素的类的前提下定义作用于这些元素的新操作。

      因才疏学浅，吐舌头，部分设计模式的解释难免有点牵强，仅供大家茶余饭后娱乐娱乐，请笑纳！微笑