前面两个ID生成器只是简单的完成功能，如果实际应用到生产环境，则对ID生成器的要求更高，具体包括但不限于以下几点：

1. 产生全局唯一、且单调递增的ID；
2. 任何情况下ID不能重复或者回退；
3. 具备高效率产生ID的能力；
4. 具备提供多种ID的能力；
5. 便于运维管理；

ID生成器整个系统需要分为四个部分：web管理端、IdGen服务、redis以及mysql，其中：

（1）web管理端用于运维人员通过网页形式管理ID生成器，它可以添加/删除ID、迁移ID等；

（2）IdGen服务属于自研服务，采用Thrift框架，对外提供各种语言的调用接口，对内管理redis和mysql，可水平扩展；

（3）Redis，在本设计中，采用redis作为Id分发器，一个redis可以分发若干类型的ID，当一个redis无法承载请求量时，可以将id类型迁移到新的Redis上。

（4）Mysql，在本设计中，采用Mysql做Id的持久化存储和配置管理，只用一个mysql即可，在实际应用中，还可以采用主从配置提升数据保存的安全性，但是从Mysql只做数据保存，不提供服务，防止主mysql所在主机出现问题时，数据可以正常恢复。



获取id的接口每调用一次，ID生成器就产生一个id，在接口中每次获取到一个id之后，都需要判断该分配的id值:curID；

如下图所示，如果curID < updateID，这里update是更新IP段时对应的ID（当前ID段损耗达到所设置比例对应的ID值），则返回该分配的ID；如果updateID < curID < endID，则要产生一个ID更新任务，如果curID >=endID，则直接返回失败。



每个IdGen都有一个线程池，用于执行异步任务，在更新ID信息的任务中，该任务包含以下信息：ID的名称，该ID所属的Redis、该ID在IdGen中缓存的对象，此任务需要计算申请ID段的大小，其方法为根据当前段的使用速度来计算：

(当前ID-起始ID)/(当前时间-本段加载时间) \* 最大加载间隔时间

如果计算的结果大于配置参数的上限，则使用上限值，反之，如果小于配置的下限值，则使用下限值。

此更新ID信息的任务将完成以下操作：

（1）从缓存中重新读取当前id的信息，并与本地保存的信息比较，如果不一致（结束ID、更新ID和更新时间都大于本地保存的值），否则说明缓存已经被其他IdGen更新，只需将本地缓存的数据（本段ID的加载时间、起始ID、结束ID、更新ID）更新为Redis中获取的数据即可；否则，说明该ID在Redis中的数据还未更新，则继续执行；

（2）在对应Redis中，对该ID加分布式锁，如果加锁失败，说明有其他的IdGen正在更新该ID的信息，直接结束当前任务即可；否则，说明当前的IdGen获取了更新该ID信息的权利，继续执行；

（3）计算新申请ID段的大小newIdLen，从数据库中申请新的ID段，提供参数包括：newIdLen、当前系统时间curSysTime，采用存储过程完成，具体需完成对数据库的以下操作：

* 如果当前ID类型不是有效，则直接返回-1；
* 将start\_id字段更新为start\_id + newIdLen;
* 将last\_range字段更新为newIdLen；
* 将last\_load\_time更新为当前系统时间；
* 返回新的start\_id

（4）IdGen根据start\_id计算该ID段的end\_id、update\_id，并更新Redis中该ID的相关信息end\_id、update\_id、当前系统时间更新为新的值，并将startid设置为当前正在分配的ID值；

（5）更新本地缓存中该ID的信息：end\_id、update\_id、start\_id和start\_time。