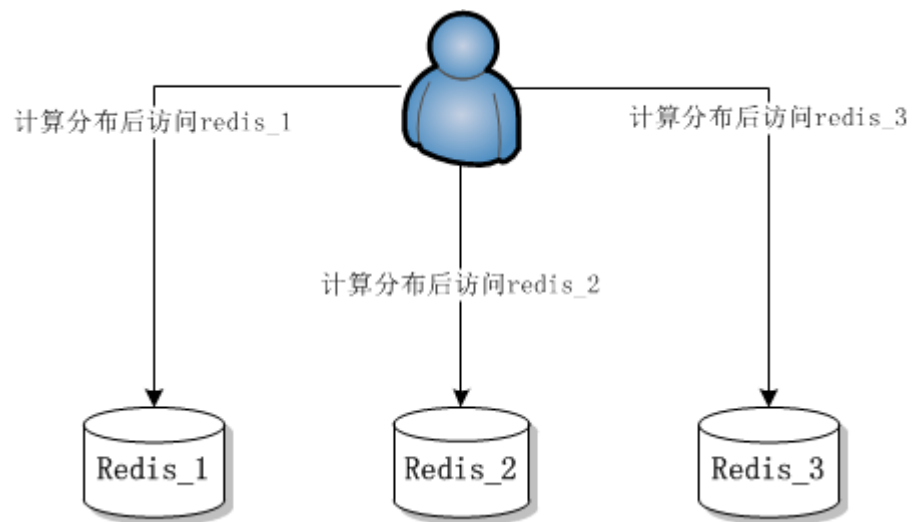


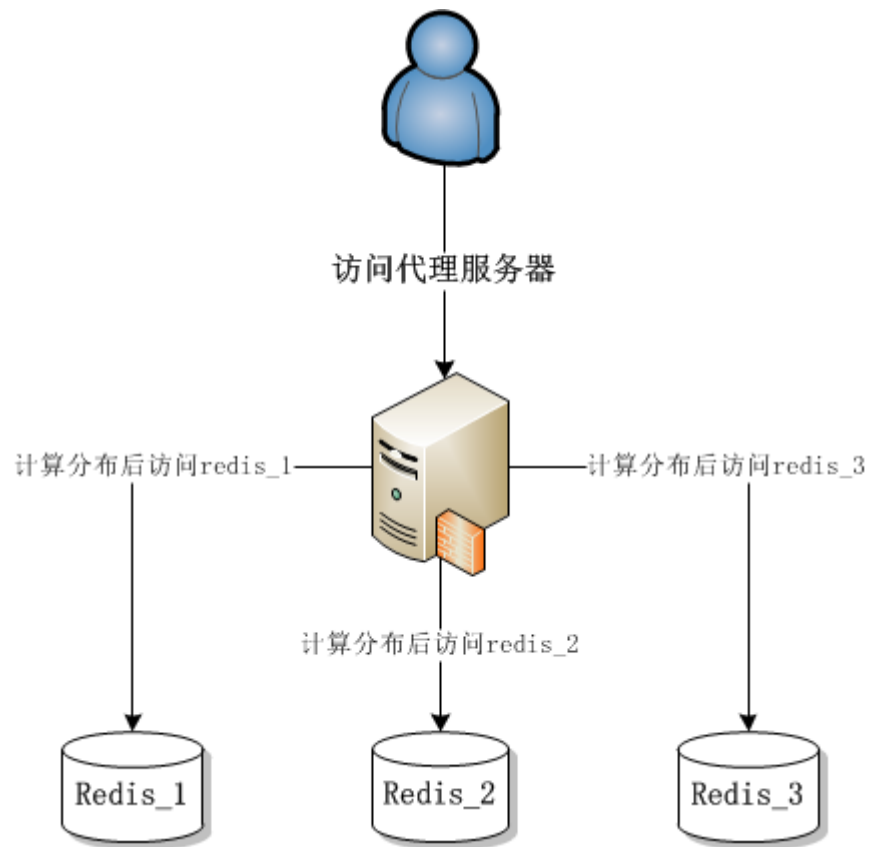
今天带给大家的是一款小工具库，在正式介绍之前，我们先讨论 redis 分布式的几种形态。

形态 1：客户端分布，请见下图：



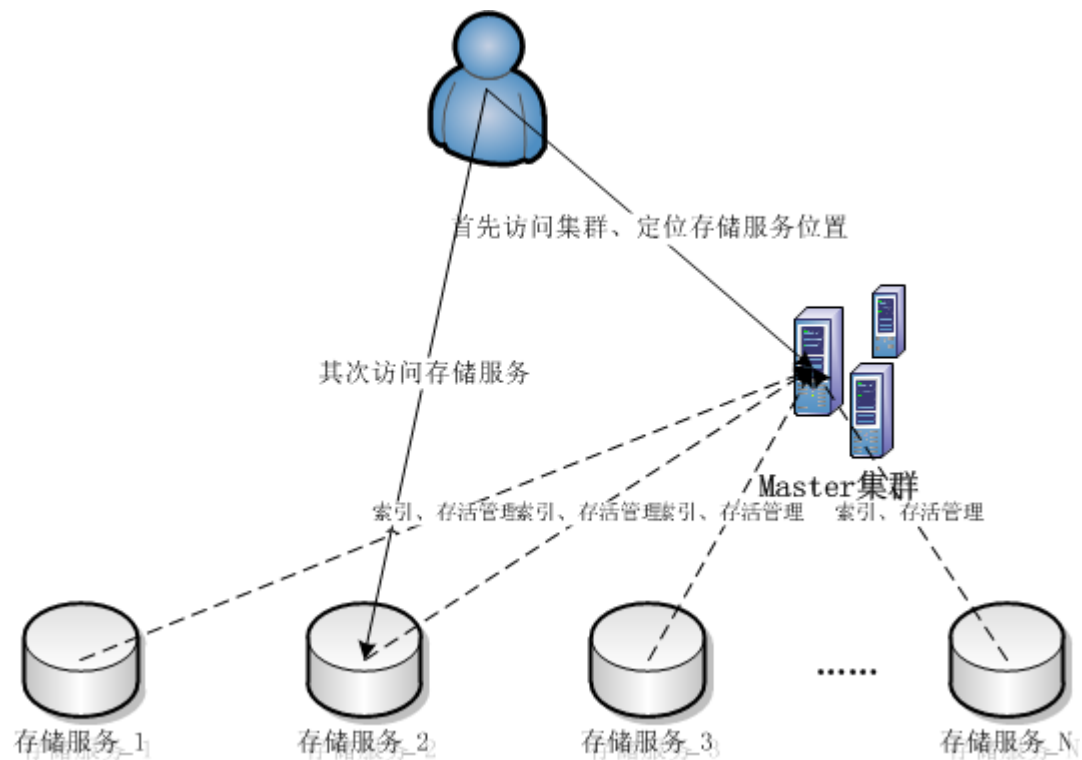
这是最常见的一种应用形态，优点是设计简单，无须后台做任何配置，redis 拿来就用。缺点是需要客户端配置分布的规则，不容易扩容，更改配置容易出错。

形态 2：服务器代理，请见下图：



与形态 1 相比,形态 2 在访问者与 redis 之间加入了一层代理服务器,用于屏蔽复杂的 redis 分布规则。YY 部分缓存应用使用了这种形态。

形态 3: 分布式缓存 (存储) , 请见下图:



在形态 2 的基础上，加入节点容错，数据备份，海量存储，以及异地多活，即分布式缓存（存储）。

Master 集群功能：

- 1) 多个 Master 之间互相选举出主服务器，当主服务器宕机或者网络故障时，其他 Master 可以重新选举替代；
- 2) 管理存储服务的存活情况，当某个存储服务发生较长时间故障时，应自动寻找备用存储服务，迁移数据；
- 3) 管理所有数据的索引，但不需要全量持久化存储；
- 4) 协调读写访问；

存储服务功能：

- 1) 管理本单元的数据存储和索引;
- 2) 启动时向 Master 集群上报索引情况;
- 3) 数据变更时向 Master 集群更新索引;
- 4) 多个存储服务之间互相同步数据;

好了,先介绍到这,这次主要看形态 1,我想文字上也不用多说了,提供的库名叫 tinyredis,也是公开的,请有需要的朋友准备好您的原创作品,并留言给微码库,只要作品验证合格,就能交换到这款源代码。

编译:

make all

在 tinyredis 目录下会生成 4 个 lib 文件,分别如下:

- 1) libtinyredis.a 单线程 debug 版本
- 2) libtinyredis_mt.a 多线程 debug 版本
- 3) libtinyredis.ra 单线程 release 版本
- 4) libtinyredis_mt.ra 多线程 release 版本

另外, sample.cpp 是使用样例,介绍了如何对单 KV、多单 V、hash 结构进行读写访问。

使用说明:

头文件:

```
#include "RedisFactory.h"
```

方法介绍:

1) 添加分布式实例:

```
void
```

```
CRedisFactory::addRedis(const std::string& strIp, uint16_t uPort16, const std::string  
& strPass, uint32_t uMiniSeconds)
```

2) 根据 key 获取实例对象:

```
CRedisClient* CRedisFactory::getRedis(uint32_t uKey);
```

```
CRedisClient* CRedisFactory::getRedis(const std::string& strKey);
```

3) 执行命令:

```
redisReply* CRedisClient::command(const char* szFormat, ...);
```

4) 访问结果集:

```
CResult::CResult(bool bAutoFree = true); 构造结果集, 参数表示是否自动释放资源
```

```
bool CResult::isArray(); 结果集是否数组
```

```
bool CResult::isInteger(); 结果集是否整形
```

```
bool CResult::isString(); 结果集是否字符串
```

```
bool CResult::isNil(); 结果集是否未命中
```

```
bool CResult::isStatus(); 结果集是否操作状态
```

```
redisReply* CResult::getSubReply(size_t uPos); 获取数组的子结果
```

size_t CResult::getArraySize(); 获取数组元素大小

int64_t CResult::getInteger(); 获取整形值

void CResult::getString(std::string& str); 获取字符串

bool CResult::isOK(); 检查操作是否成功

以下是从 sample.cpp 提取的单 KV 读写访问示例代码：

```
using namespace tinyredis;
```

```
// 创建工厂实例
```

```
CRedisFactory redisFactory;
```

```
// 初使化 redis 服务配置
```

```
redisFactory.addRedis("127.0.0.1", 3000, "123456", 1000);
```

```
redisFactory.addRedis("127.0.0.1", 3001, "123456", 1000);
```

```
uint32_t uld = 1;
```

```
std::string strName = "zhang3";
```

```
CRedisClient* pRedis = redisFactory.getRedis(uld);
```

```
CResult result(true);
```

```
result = pRedis->command("set name:%u %s", uld, strName.c_str());
```

```
if (!result)
{
    printf("set failed : %s \n", pRedis->getErrStr().c_str());

    return;
}
```

```
printf("set ok \n");
```

```
result = pRedis->command("get name:%u", uid);

if (!result)
{
    printf("get failed : %s \n", pRedis->getErrStr().c_str());

    return;
}
```

```
if (result.isNil())
{
    printf("not found! \n");

    return;
}
```

```
if (result.isString())
```

```
{  
    std::string strValue;  
    result.getString(strValue);  
    printf("get value : %s \n", strValue.c_str());  
}
```