多位业界专家联合推荐
来自一线开发者的实战经验总结

Redis 入门指南

（第 2 版）

李子骅 编著

真正**零基础**入门，**深入浅出**全面剖析 Redis
任务驱动式学习，**轻松掌握** Redis 实战知识

人民邮电出版社
POSTS & TELECOM PRESS

Redis 入门指南（第 2 版）

李子骅　编著

人 民 邮 电 出 版 社
北　京

内 容 提 要

本书是一本 Redis 的入门指导书籍，以通俗易懂的方式介绍了 Redis 基础与实践方面的知识，包括历史与特性、在开发和生产环境中部署运行 Redis、数据类型与命令、使用 Redis 实现队列、事务、复制、管道、持久化、优化 Redis 存储空间等内容，并采用任务驱动的方式介绍了 PHP、Ruby、Python 和 Node.js 这 4 种语言的 Redis 客户端库的使用方法。

本书的目标读者不仅包括 Redis 的新手，还包括那些已经掌握 Redis 使用方法的人。对于新手而言，本书的内容由浅入深且紧贴实践，旨在让读者真正能够即学即用；对于已经了解 Redis 的读者，通过本书的大量实例以及细节介绍，也能发现很多新的技巧。

◆ 编　　著　李子骅
责任编辑　杨海玲
责任印制　张佳莹

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：14
字数：296 千字　　2015 年 5 月第 2 版
印数：1 – 0 000 册　　2015 年 5 月北京第 1 次印刷

定价：00.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京崇工商广字第 0021 号

前　言

Redis 如今已经成为 Web 开发社区中最火热的内存数据库之一，而它的诞生距现在不过才 4 年。随着 Web 2.0 的蓬勃发展，网站数据快速增长，对高性能读写的需求也越来越多，再加上半结构化的数据比重逐渐变大，人们对早已被铺天盖地地运用着的关系数据库能否适应现今的存储需求产生了疑问。而 Redis 的迅猛发展，为这个领域注入了全新的思维。

Redis 凭借其全面的功能得到越来越多的公司的青睐，从初创企业到新浪微博这样拥有着几百台 Redis 服务器的大公司，都能看到 Redis 的身影。Redis 也是一个名副其实的多面手，无论是存储、队列还是缓存系统，都有它的用武之地。

本书将从 Redis 的历史讲起，结合基础与实践，带领读者一步步进入 Redis 的世界。

第 2 版说明

在本书第 1 版截稿的时候，加入了 Lua 脚本功能的 Redis 2.6 版刚刚发布，此时的 Redis 正在逐渐地被国内的开发者所熟知。如今整整两年过去了，Redis 也即将发布新的里程碑版本 3.0 版。在这两年中，Redis 增加了许多优秀的功能，同时也被越来越多的公司所采用与信赖。在写这段文字时，恰好 Redis 的作者 Salvatore Sanfilippo 转述了别人的一句话：“如果把 Redis 官网的‘谁在使用 Redis’页面改名为‘谁没在使用 Redis’，那么这个页面的内容一定会精简不少。”虽然是一句玩笑话，但是也从侧面体现出这两年里 Redis 的飞速发展。而继续编写《Redis 入门指南》第 2 版的最大动力也是希望将 Redis 发展的成果及时地与广大读者分享，同时也借此感谢大家对本书第 1 版的积极反馈。

目标读者

本书假定读者是Redis的新手，甚至可能连Redis是什么都没听说过。本书将会详细地介绍Redis是什么以及为什么要使用 Redis，旨在能让读者从零开始逐步晋升为一个优秀的 Redis 开发者。

本书还包含了很多 Redis 实践方面的知识，对于有经验的 Redis 开发者，大可以直接跳过已经掌握的内容，只阅读感兴趣的部分。每章的引言都简要介绍了本章要讲解的内容，供读者参考。

本书并不需要读者有任何 Redis 的背景知识，不过如果读者有 Web 后端开发经验或 Linux 系统使用经验，阅读本书将会更加得心应手。

组织结构

第 1 章介绍了 Redis 的历史与特性，主要回答两个初学者最关心的问题，即 Redis 是什么和为什么要使用 Redis。

第 2 章讲解了如何安装和运行 Redis。如果你身旁的计算机没有运行 Redis，那么一定不要错过这一章，因为本书后面的部分都需要读者最好能一边阅读一边实践，以提高学习效率。本章中还会介绍 Redis 命令行客户端的使用方法等基础知识，这些都是实践前需要掌握的知识。

第 3 章介绍了 Redis 的数据类型。本章讲解的不仅是每个数据类型的介绍和命令的格式，还会着重讲解每个数据类型分别在实践中如何使用。整个第 3 章会带领读者从零开始，一步步地使用 Redis 构建一个博客系统，旨在帮助读者在学习完本章的内容之后可以直接在自己的项目中上手实践 Redis。

第 4 章引入了一些 Redis 的进阶知识，比如事务和消息系统等。同样本章还会继续以博客系统为例子，以实践驱动学习。

第 5 章介绍了如何在各个编程语言中使用 Redis，这些语言包括 PHP、Ruby、Python 和 Node.js。其中讲解每种语言时最后都会以一个有趣的例子作为演示，即使你不了解某些语言，阅读这些例子也能让你收获颇丰。

第 6 章展示了 Redis 脚本的强大功能。本章会向读者讲解如何借助脚本来扩展 Redis，并且会对脚本一些需要注意的地方（如沙盒、随机结果等）进行着重介绍。

第 7 章会介绍 Redis 持久化的知识。Redis 持久化包含 RDB 和 AOF 两种方式，对持久化的支持是 Redis 之所以可以用作数据库的必要条件。

第 8 章详细说明了多个 Redis 实例的维护方法，包括使用复制实现读写分离、借助哨兵来自动完成故障恢复以及通过集群来实现数据分片。

第 9 章介绍了 Redis 安全和协议相关的内容，并向会推荐几个第三方的 Redis 管理工具。

附录 A 收录了 Redis 命令的不同属性，以及属性的特征。

附录 B 收录了 Redis 部分配置参数的章节索引。

附录 C 收录了 Redis 使用的 CRC16 实现代码。

排版约定

本书排版使用字体遵从以下约定。

- 等宽字：表示在命令行中输入的命令以及返回结果、程序代码、Redis 的命令（包括命令语句和命令定义）。
- 等宽斜体字（或夹在其中的中文楷体字）：表示命令或程序代码中由读者自行替换的参数或变量。

- 等宽粗体字：表示命令行中用户的输入内容、伪代码中的 Redis 命令。
- 命令行的输入和输出以如下格式显示：

```
$ redis-cli PING
PONG
```

- Redis 命令行客户端的输入和输出以如下格式显示：

```
redis> SET foo bar
OK
```

- 程序代码以如下格式显示：

```
var redis = require("redis");
var client = redis.createClient();

// 将两个对象 JSON 序列化后存入数据库中
client.mset(
    'user:1', JSON.stringify(bob),
    'user:2', JSON.stringify(jeff)
);
```

代码约定

本书的部分章节采用了伪代码来讲解，这种伪代码类似 Ruby 和 PHP，例如：

```
def hsetnx($key, $field, $value)
    $isExists = HEXISTS $key, $field
    if $isExists is 0
        HSET $key, $field, $value
        return 1
    else
        return 0
```

其中变量使用$符号标识，Redis 命令使用的粗体表示并省略了括号以便于阅读。在命令调用和 print 等语句中没有$符号的字符串会被当作字符串字面值。

附加文件

本书第 5 章中每一节都包含了一个完整的程序，通常来讲读者最好自己输入这些代码来加深理解，当然如果要先看到程序的运行结果再开始学习也不失为一个好办法。

这些程序代码都存放在 GitHub 上（https://github.com/luin/redis-book-assets），可以在 GitHub 上查看与下载。

致谢

在本书写作的过程中，得到了很多朋友的帮助。请允许我在这里占用少许篇幅，向他们致以诚挚的谢意。

感谢人民邮电出版社的杨海玲老师对本书的支持，没有她的悉心指导，本书就无法顺利完成。

感谢刘亚晨、李欣越、寇祖阳和余尧，他们承担了许多额外的工作，使得我可以全身心地投入到写作中。

感谢所有浏览本书初稿并提出意见和建议的人们：张沈鹏、陈硕实、刘其帅、扈煊、李其超、朱冲宇、王诗吟、黄山月、刘昕、韩重远、李申申、杨海朝、田琪等。感谢你们的支持。

另外还要感谢“宋老师”，是的，就是书中的主人公之一。几年前，我刚创业时，办公场所是和某个教育机构合租的，宋老师是该机构的一名英语老师，同时他也是国内一个知名的嘻哈乐团成员之一。他平日风趣的谈吐带给了我们很多欢乐，伴随我们走过了艰苦的创业初期，而我接触 Redis，也正是从这段时间开始的。

最后，感谢我的父母和女朋友马梦妍，你们是我生命中最重要的人，感谢你们的理解和支持。

目 录

第 1 章

简介

Redis 是一个开源的、高性能的、基于键值对的缓存与存储系统，通过提供多种键值数据类型来适应不同场景下的缓存与存储需求。同时 Redis 的诸多高层级功能使其可以胜任消息队列、任务队列等不同的角色。

本章将分别介绍 Redis 的历史和特性，以使读者能够快速地对 Redis 有一个全面的了解。

1.1 历史与发展

2008 年，意大利的一家创业公司 Merzia[①]推出了一款基于 MySQL 的网站实时统计系统 LLOOGG[②]，然而没过多久该公司的创始人 Salvatore Sanfilippo 便开始对 MySQL 的性能感到失望，于是他决定亲自为 LLOOGG 量身定做一个数据库，并于 2009 年开发完成，这个数据库就是 Redis。不过 Salvatore Sanfilippo 并不满足只将 Redis 用于 LLOOGG 这一款产品，而是希望让更多的人使用它，于是在同一年 Salvatore Sanfilippo 将 Redis 开源发布，并开始和 Redis 的另一名主要的代码贡献者 Pieter Noordhuis 一起继续着 Redis 的开发，直到今天。

Salvatore Sanfilippo 自己也没有想到，短短的几年时间，Redis 就拥有了庞大的用户群体。Hacker News 在 2012 年发布了一份数据库的使用情况调查[③]，结果显示有近 12%的公司在使用 Redis。国内如新浪微博、街旁和知乎，国外如 GitHub、Stack Overflow、Flickr、暴雪和 Instagram，都是 Redis 的用户。

VMware 公司从 2010 年开始赞助 Redis 的开发，Salvatore Sanfilippo 和 Pieter Noordhuis

① http://merzia.com
② http://lloogg.com
③ http://news.ycombinator.com/item?id=4833188

也分别于同年的 3 月和 5 月加入 VMware，全职开发 Redis。

Redis 的代码托管在 GitHub 上，开发十分活跃[①]。2015 年 4 月 2 日，Redis 发布了 3.0.0 的正式版本。

1.2　特性

作为一款个人开发的系统，Redis 究竟有什么魅力吸引了如此多的用户呢？

1.2.1　存储结构

有过脚本语言编程经验的读者对字典（或称映射、关联数组）数据结构一定很熟悉，如代码 `dict["key"] = "value"`中 `dict` 是一个字典结构变量，字符串`"key"`是键名，而`"value"`是键值，在字典中我们可以获取或设置键名对应的键值，也可以删除一个键。

Redis 是 REmote DIctionary Server（远程字典服务器）的缩写，它以字典结构存储数据，并允许其他应用通过 TCP 协议读写字典中的内容。同大多数脚本语言中的字典一样，Redis 字典中的键值除了可以是字符串，还可以是其他数据类型。到目前为止 Redis 支持的键值数据类型如下：

- 字符串类型
- 散列类型
- 列表类型
- 集合类型
- 有序集合类型

这种字典形式的存储结构与常见的 MySQL 等关系数据库的二维表形式的存储结构有很大的差异。举个例子，如下所示，我们在程序中使用 post 变量存储了一篇文章的数据（包括标题、正文、阅读量和标签）：

```
post["title"]  =  "Hello World!"
post["content"] = "Blablabla..."
post["views"] = 0
post["tags"] = ["PHP", "Ruby", "Node.js"]
```

现在我们希望将这篇文章的数据存储在数据库中，并且要求可以通过标签检索出文章。如果使用关系数据库存储，一般会将其中的标题、正文和阅读量存储在一个表中，而将标签存储在另一个表中，然后使用第三个表连接文章和标签表[②]。需要查询时还得将 3 个表进

① https://github.com/antirez/redis

② 这是一种符合第三范式的设计。事实上还可以使用其他方式来实现标签系统，参阅（http://tagging.pui.ch/post/37027745720/ tags-database-schemas）以了解更多相关资料。

行连接，不是很直观。而 Redis 字典结构的存储方式和对多种键值数据类型的支持使得开发者可以将程序中的数据直接映射到 Redis 中，数据在 Redis 中的存储形式和其在程序中的存储方式非常相近。使用 Redis 的另一个优势是其对不同的数据类型提供了非常方便的操作方式，如使用集合类型存储文章标签，Redis 可以对标签进行如交集、并集这样的集合运算操作。3.5 节会专门介绍如何借助集合运算轻易地实现“找出所有同时属于 A 标签和 B 标签且不属于 C 标签”这样关系数据库实现起来性能不高且较为繁琐的操作。

1.2.2 内存存储与持久化

Redis 数据库中的所有数据都存储在内存中。由于内存的读写速度远快于硬盘，因此 Redis 在性能上对比其他基于硬盘存储的数据库有非常明显的优势，在一台普通的笔记本电脑上，Redis 可以在一秒内读写超过 10 万个键值。

将数据存储在内存中也有问题，比如程序退出后内存中的数据会丢失。不过 Redis 提供了对持久化的支持，即可以将内存中的数据异步写入到硬盘中，同时不影响继续提供服务。

1.2.3 功能丰富

Redis 虽然是作为数据库开发的，但由于其提供了丰富的功能，越来越多的人将其用作缓存、队列系统等。Redis 可谓是名副其实的多面手。

Redis 可以为每个键设置生存时间（Time To Live，TTL），生存时间到期后键会自动被删除。这一功能配合出色的性能让 Redis 可以作为缓存系统来使用，而且由于 Redis 支持持久化和丰富的数据类型，使其成为了另一个非常流行的缓存系统 Memcached 的有力竞争者。

> **讨论** 关于 Redis 和 Memcached 优劣的讨论一直是一个热门的话题。在性能上 Redis 是单线程模型，而 Memcached 支持多线程，所以在多核服务器上后者的性能理论上相对更高一些。然而，前面已经介绍过，Redis 的性能已经足够优异，在绝大部分场合下其性能都不会成为瓶颈，所以在使用时更应该关心的是二者在功能上的区别。随着 Redis 3.0 的推出，标志着 Memcached 几乎所有功能都成为了 Redis 的子集。同时，Redis 对集群的支持使得 Memcached 原有的第三方集群工具不再成为优势。因此，在新项目中使用 Redis 代替 Memcached 将会是非常好的选择。

作为缓存系统，Redis 还可以限定数据占用的最大内存空间，在数据达到空间限制后可以按照一定的规则自动淘汰不需要的键。

除此之外，Redis 的列表类型键可以用来实现队列，并且支持阻塞式读取，可以很容易地实现一个高性能的优先级队列。同时在更高层面上，Redis 还支持“发布/订阅”的消息模式，可以基于此构建聊天室[①]等系统。

1.2.4　简单稳定

即使功能再丰富，如果使用起来太复杂也很难吸引人。Redis 直观的存储结构使得通过程序与 Redis 交互十分简单。在 Redis 中使用命令来读写数据，命令语句之于 Redis 就相当于 SQL 语言之于关系数据库。例如在关系数据库中要获取 posts 表内 id 为 1 的记录的 title 字段的值可以使用如下 SQL 语句实现：

```
SELECT title FROM posts WHERE id = 1 LIMIT 1
```

相对应的，在 Redis 中要读取键名为 `post:1` 的散列类型键的 title 字段的值，可以使用如下命令语句实现：

```
HGET post:1 title
```

其中 `HGET` 就是一个命令。Redis 提供了 100 多个命令（如图 1-1 所示），听起来很多，但是常用的却只有十几个，并且每个命令都很容易记忆。读完第 3 章你就会发现 Redis 的命令比 SQL 语言要简单很多。

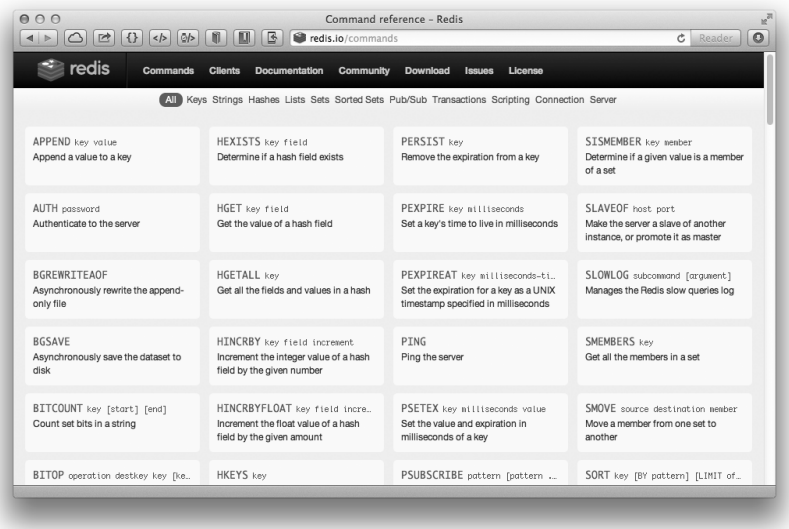

图 1-1　Redis 官网提供了详细的命令文档

① Redis 的贡献者之一 Pieter Noordhuis 提供了一个使用该模式开发的聊天室的例子，见 https://gist.github.com/348262。

Redis 提供了几十种不同编程语言的客户端库，这些库都很好地封装了 Redis 的命令，使得在程序中与 Redis 进行交互变得更容易。有些库还提供了可以将编程语言中的数据类型直接以相应的形式存储到 Redis 中（如将数组直接以列表类型存入 Redis）的简单方法，使用起来非常方便。

Redis 使用 C 语言开发，代码量只有 3 万多行。这降低了用户通过修改 Redis 源代码来使之更适合自己项目需要的门槛。对于希望“榨干”数据库性能的开发者而言，这无疑是一个很大的吸引力。

Redis 是开源的，所以事实上 Redis 的开发者并不止 Salvatore Sanfilippo 和 Pieter Noordhuis。截至目前，有将近 100 名开发者为 Redis 贡献了代码。良好的开发氛围和严谨的版本发布机制使得 Redis 的稳定版本非常可靠，如此多的公司在项目中使用了 Redis 也可以印证这一点。

第 2 章

准备

“纸上得来终觉浅，绝知此事要躬行。”

——陆游《冬夜读书示子聿》

学习 Redis 最好的办法就是动手尝试它。在介绍 Redis 最核心的内容之前，本章先来介绍一下如何安装和运行 Redis，以及 Redis 的基础知识，使读者可以在之后的章节中一边学习一边实践。

2.1 安装 Redis

安装 Redis 是开始 Redis 学习之旅的第一步。在安装 Redis 前需要了解 Redis 的版本规则以选择最适合自己的版本，Redis 约定次版本号（即第一个小数点后的数字）为偶数的版本是稳定版（如 2.8 版、3.0 版），奇数版本是非稳定版（如 2.7 版、2.9 版），生产环境下一般需要使用稳定版本。本书的内容以 3.0 版为目标编写，同时绝大部分内容也适用于 2.6 版和 2.8 版。对于只在最新版才有的特性（如 Cluster 集群），本书会做特别说明。

2.1.1 在 POSIX 系统中安装

Redis 兼容大部分 POSIX 系统，包括 Linux、OS X 和 BSD 等，在这些系统中推荐直接下载 Redis 源代码编译安装以获得最新的稳定版本。Redis 最新稳定版本的源代码可以从地址 http://download.redis.io/redis-stable.tar.gz 下载。

下载安装包后解压即可使用 `make` 命令完成编译，完整的命令如下：

```
wget http://download.redis.io/redis-stable.tar.gz
tar xzf redis-stable.tar.gz
cd redis-stable
make
```

Redis 没有其他外部依赖，安装过程很简单。编译后在 Redis 源代码目录的 src 文件夹中可以找到若干个可执行程序，最好在编译后直接执行 make install 命令来将这些可执行程序复制到/usr/local/bin 目录中以便以后执行程序时可以不用输入完整的路径。

在实际运行 Redis 前推荐使用 make test 命令测试 Redis 是否编译正确，尤其是在编译一个不稳定版本的 Redis 时。

> **提示**　除了手工编译外，还可以使用操作系统中的软件包管理器来安装 Redis，但目前大多数软件包管理器中的 Redis 的版本都较古老。考虑到 Redis 的每次升级都提供了对以往版本的问题修复和性能提升，使用最新版本的 Redis 往往可以提供更加稳定的体验。如果希望享受包管理器带来的便利，在安装前请确认您使用的软件包管理器中 Redis 的版本并了解该版本与最新版之间的差异。http://redis.io/topics/problems 中列举了一些在以往版本中存在的已知问题。

2.1.2　在 OS X 系统中安装

OS X 下的软件包管理工具 Homebrew 和 MacPorts 均提供了较新版本的 Redis 包，所以我们可以直接使用它们来安装 Redis，省去了像其他 POSIX 系统那样需要手动编译的麻烦。下面以使用 Homwbrew 安装 Redis 为例。

1. 安装 Homebrew

在终端下输入 ruby -e "$(curl -fsSkL raw.github.com/mxcl/homebrew/go)" 即可安装 Homebrew。

如果之前安装过 Homebrew，请执行 brew update 来更新 Homebrew，以便安装较新版的 Redis。

2. 通过 Homebrew 安装 Redis

使用 brew install 软件包名可以安装相应的包，此处执行 brew install redis 来安装 Redis：

```
$ brew install redis
```

```
==> Downloading
https://downloads.sf.net/project/machomebrew/Bottles/redis-3.0.0.yosemite.bottle
.tar.gz
######################################################################## 100.0%
==> Pouring redis-3.0.0.yosemite.bottle.tar.gz
==> Caveats
To have launchd start redis at login:
    ln -sfv /usr/local/opt/redis/*.plist ~/Library/LaunchAgents
Then to load redis now:
    launchctl load ~/Library/LaunchAgents/homebrew.mxcl.redis.plist
Or, if you don't want/need launchctl, you can just run:
    redis-server /usr/local/etc/redis.conf
==> Summary
    /usr/local/Cellar/redis/3.0.0: 10 files, 1.4M
```

OS X 系统从 Tiger 版本开始引入了 launchd 工具来管理后台程序，如果想让 Redis 随系统自动运行可以通过以下命令配置 launchd：

```
ln -sfv /usr/local/opt/redis/*.plist ~/Library/LaunchAgents
launchctl load ~/Library/LaunchAgents/homebrew.mxcl.redis.plist
```

通过 launchd 运行的 Redis 会加载位于/usr/local/etc/redis.conf 的配置文件，关于配置文件会在 2.4 节中介绍。

2.1.3 在 Windows 中安装

Redis 官方不支持 Windows。2011 年微软[①]向 Redis 提交了一个补丁，以使 Redis 可以在 Windows 下编译运行，但被 Salvatore Sanfilippo 拒绝了，原因是在服务器领域上 Linux 已经得到了广泛的使用，让 Redis 能在 Windows 下运行相比而言显得不那么重要。并且 Redis 使用了如写时复制等很多操作系统相关的特性，兼容 Windows 会耗费太大的精力而影响 Redis 其他功能的开发。尽管如此微软还是发布了一个可以在 Windows 运行的 Redis 分支[②]，而且更新相当频繁，截止到本书交稿时，Windows 下的 Redis 版本为 2.8。

如果想使用 Windows 学习或测试 Redis 可以通过 Cygwin 软件或虚拟机（如 VirtualBox）来完成。Cygwin 能够在 Windows 中模拟 Linux 系统环境。Cygwin 实现了一个 Linux API 接口，使得大部分 Linux 下的软件可以重新编译后在 Windows 下运行。Cygwin 还提供了自己的软件包管理工具，让用户能够方便地安装和升级几千个软件包。借助 Cygwin，我们可以在 Windows 上通过源代码编译安装最新版的 Redis。

① 微软开放技术有限公司（Microsoft Open Technologies Inc.），专注于参与开源项目、开放标准工作组以及提出倡议。
② https://github.com/MSOpenTech/Redis

1. 安装 Cygwin

从 Cygwin 官方网站（http://cygwin.com）下载 setup.exe 程序，setup.exe 既是 Cygwin 的安装包，又是 Cygwin 的软件包管理器。运行 setup.exe 后进入安装向导。前几步会要求选择下载源、安装路径、代理和下载镜像等，可以根据具体需求选择，一般来说一路点击“Next”即可。之后会出现软件包管理界面，如图 2-1 所示。

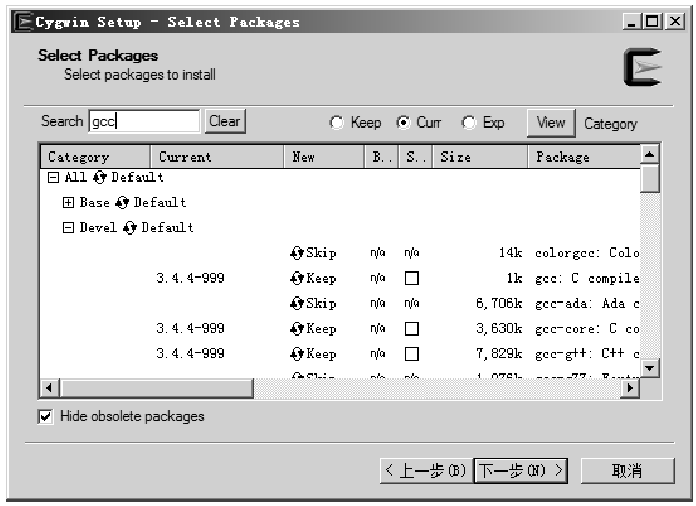

图 2-1　Cygwin 包管理界面

编译安装 Redis 需要用到的包有 gcc 和 make，二者都可以在“Devel”分类中找到。在“New”字段中标记为“Skip”的包表示不安装，单击“Skip”切换成需要安装的版本号即可令 Cygwin 在稍后安装该版本的包。图 2-1 中所示 gcc 包的状态为“Keep”是因为作者之前已经安装过该包了，同样如果读者在退出安装向导后还想安装其他软件包，只需要重新运行 setup.exe 程序再次进入此界面即可。

为了方便使用，我们还可以安装 wget（用于下载 Redis 源代码，也可以手动下载并使用 Windows 资源管理器将其复制到 Cygwin 对应的目录中，见下文介绍）和 vim（用于修改 Redis 的源代码使之可以在 Cygwin 下正常编译）。

之后单击“Next”，安装向导就会自动完成下载和安装工作了。

安装成功后打开 Cygwin Terminal 程序即可进入 Cygwin 环境，Cygwin 会将 Windows 中的目录映射到 Cygwin 中。如果安装时没有更改安装目录，Cygwin 环境中的根目录对应的 Windows 中的目录是 C:\cygwin。

2. 修改 Redis 源代码

下载和解压 Redis 的过程和 2.1.1 节中介绍的一样，不过在 `make` 之前还需要修改 Redis

的源代码以使其可以在 Cygwin 下正常编译。

首先编辑 src 目录下的 redis.h 文件，在头部加入：

```
#ifdef CYGWIN
#ifndef SA ONSTACK
#define SA ONSTACK 0x08000000
#endif
#endif
```

而后编辑 src 目录下的 object.c 文件，在头部加入：

```
#define strtold(a,b) ((long double)strtod((a),(b)))
```

3. 编译 Redis

同 2.1.1 节一样，执行 `make` 命令即可完成编译。

> **注意** Cygwin 环境无法完全模拟 Linux 系统，比如 Cygwin 的 fork 不支持写时复制；另外，Redis 官方也并不提供对 Cygwin 的支持，Cygwin 环境只能用于学习 Redis。运行 Redis 的最佳系统是 Linux 和 OS X，官方推荐的生产系统是 Linux。

2.2 启动和停止 Redis

安装完 Redis 后的下一步就是启动它，本节将分别介绍在开发环境和生产环境中运行 Redis 的方法以及正确停止 Redis 的步骤。

在这之前首先需要了解 Redis 包含的可执行文件都有哪些，表 2-1 中列出了这些程序的名称以及对应的说明。如果在编译后执行了 `make install` 命令，这些程序会被复制到 /usr/local/bin 目录内，所以在命令行中直接输入程序名称即可执行。

表 2-1 Redis 可执行文件说明

文 件 名	说 明
redis-server	Redis 服务器
redis-cli	Redis 命令行客户端
redis-benchmark	Redis 性能测试工具
redis-check-aof	AOF 文件修复工具
redis-check-dump	RDB 文件检查工具
redis-sentinel	Sentinel 服务器（仅在 2.8 版以后）

我们最常使用的两个程序是 redis-server 和 redis-cli，其中 redis-server 是 Redis 的服务器，启动 Redis 即运行 redis-server；而 redis-cli 是 Redis 自带的 Redis 命令行客户端，是学习 Redis 的重要工具，2.3 节会详细介绍它。

2.2.1　启动 Redis

启动 Redis 有直接启动和通过初始化脚本启动两种方式，分别适用于开发环境和生产环境。

1. 直接启动

直接运行 redis-server 即可启动 Redis，十分简单：

```
$ redis-server
[5101] 14 Dec 20:58:59.944 # Warning: no config file specified, using the default
config. In order to specify a config file use redis-server /path/to/redis.conf
[5101] 14 Dec 20:58:59.948 * Max number of open files set to 10032

...

[5101] 14 Dec 20:58:59.949 # Server started, Redis version 2.6.9
[5101] 14 Dec 20:58:59.949 * The server is now ready to accept connections on port 6379
```

Redis 服务器默认会使用 6379 端口[①]，通过--port 参数可以自定义端口号：

```
$ redis-server --port 6380
```

2. 通过初始化脚本启动 Redis

在 Linux 系统中可以通过初始化脚本启动 Redis，使得 Redis 能随系统自动运行，在生产环境中推荐使用此方法运行 Redis，这里以 Ubuntu 和 Debian 发行版为例进行介绍。在 Redis 源代码目录的 utils 文件夹中有一个名为 redis_init_script 的初始化脚本文件，内容如下：

```
#!/bin/sh
#
# Simple Redis init.d script conceived to work on Linux systems
# as it does use of the /proc filesystem.

REDISPORT=6379
EXEC=/usr/local/bin/redis-server
CLIEXEC=/usr/local/bin/redis-cli

PIDFILE=/var/run/redis_${REDISPORT}.pid
```

① 6379 是手机键盘上 MERZ 对应的数字，MERZ 是一名意大利歌女的名字。

```
CONF="/etc/redis/${REDISPORT}.conf"

case "$1" in
    start)
        if [ -f $PIDFILE ]
        then
                echo "$PIDFILE exists, process is already running or crashed"
        else
                echo "Starting Redis server..."
                $EXEC $CONF
        fi
        ;;
    stop)
        if [ ! -f $PIDFILE ]
        then
                echo "$PIDFILE does not exist, process is not running"
        else
                PID=$(cat $PIDFILE)
                echo "Stopping ..."
                $CLIEXEC -p $REDISPORT shutdown
                while [ -x /proc/${PID} ]
                do
                    echo "Waiting for Redis to shutdown ..."
                    sleep 1
                done
                echo "Redis stopped"
        fi
        ;;
    *)
        echo "Please use start or stop as first argument"
        ;;
esac
```

我们需要配置 Redis 的运行方式和持久化文件、日志文件的存储位置等，具体步骤如下。

（1）配置初始化脚本。首先将初始化脚本复制到/etc/init.d 目录中，文件名为 redis_端口号，其中端口号表示要让 Redis 监听的端口号，客户端通过该端口连接 Redis。然后修改脚本第 6 行的 REDISPORT 变量的值为同样的端口号。

（2）建立需要的文件夹。建立表 2-2 中列出的目录。

表 2-2 需要建立的目录及说明

目 录 名	说 明
/etc/redis	存放 Redis 的配置文件
/var/redis/端口号	存放 Redis 的持久化文件

（3）修改配置文件。首先将配置文件模板（见 2.4 节介绍）复制到/etc/redis 目录中，以端口号命名（如“6379.conf”），然后按照表 2-3 对其中的部分参数进行编辑。

表 2-3　需要修改的配置及说明

参　数	值	说　明
daemonize	yes	使 Redis 以守护进程模式运行
pidfile	/var/run/redis_端口号.pid	设置 Redis 的 PID 文件位置
port	端口号	设置 Redis 监听的端口号
dir	/var/redis/端口号	设置持久化文件存放位置

现在就可以使用/etc/init.d/redis_端口号 start 来启动 Redis 了，而后需要执行下面的命令使 Redis 随系统自动启动：

```
$ sudo update-rc.d redis_端口号 defaults
```

2.2.2　停止 Redis

考虑到 Redis 有可能正在将内存中的数据同步到硬盘中，强行终止 Redis 进程可能会导致数据丢失。正确停止 Redis 的方式应该是向 Redis 发送 SHUTDOWN 命令，方法为：

```
$ redis-cli SHUTDOWN
```

当 Redis 收到 SHUTDOWN 命令后，会先断开所有客户端连接，然后根据配置执行持久化，最后完成退出。

Redis 可以妥善处理 SIGTERM 信号，所以使用 kill *Redis 进程的 PID* 也可以正常结束 Redis，效果与发送 SHUTDOWN 命令一样。

2.3　Redis 命令行客户端

还记得我们刚才编译出来的 redis-cli 程序吗？redis-cli（Redis Command Line Interface）是 Redis 自带的基于命令行的 Redis 客户端，也是我们学习和测试 Redis 的重要工具，本书后面会使用它来讲解 Redis 各种命令的用法。

本节将会介绍如何通过 redis-cli 向 Redis 发送命令，并且对 Redis 命令返回值的不同类型进行简单介绍。

2.3.1　发送命令

通过 redis-cli 向 Redis 发送命令有两种方式，第一种方式是将命令作为 redis-cli 的参数执行，比如在 2.2.2 节中用过的 redis-cli SHUTDOWN。redis-cli 执行时会自动按照默认配

置（服务器地址为 127.0.0.1，端口号为 6379）连接 Redis，通过-h 和-p 参数可以自定义地址和端口号：

```
$ redis-cli -h 127.0.0.1 -p 6379
```

Redis 提供了 PING 命令来测试客户端与 Redis 的连接是否正常，如果连接正常会收到回复 PONG。如：

```
$ redis-cli PING
PONG
```

第二种方式是不附带参数运行 redis-cli，这样会进入交互模式，可以自由输入命令，例如：

```
$ redis-cli
redis 127.0.0.1:6379> PING
PONG
redis 127.0.0.1:6379> ECHO hi
"hi"
```

这种方式在要输入多条命令时比较方便，也是本书中主要采用的方式。为了简便起见，后文中我们将用 redis>表示 redis 127.0.0.1:6379>。

2.3.2 命令返回值

在大多数情况下，执行一条命令后我们往往会关心命令的返回值，如 1.2.4 节中的 HGET 命令的返回值就是我们需要的指定键的 title 字段的值。命令的返回值有 5 种类型，对于每种类型 redis-cli 的展现结果都不同，下面分别说明。

1. 状态回复

状态回复（status reply）是最简单的一种回复，比如向 Redis 发送 SET 命令设置某个键的值时，Redis 会回复状态 OK 表示设置成功。另外之前演示的对 PING 命令的回复 PONG 也是状态回复。状态回复直接显示状态信息，如：

```
redis> PING
PONG
```

2. 错误回复

当出现命令不存在或命令格式有错误等情况时 Redis 会返回错误回复（error reply）。错误回复以(error)开头，并在后面跟上错误信息。如执行一个不存在的命令：

```
redis> ERRORCOMMEND
```

```
(error) ERR unknown command 'ERRORCOMMEND'
```

在 2.6 版本时，错误信息均是以“ERR”开头，而在 2.8 版以后，部分错误信息会以具体的错误类型开头，如：

```
redis> LPUSH key 1
(integer) 1
redis> GET key
(error) WRONGTYPE Operation against a key holding the wrong kind of value
```

这里错误信息开头的“WRONGTYPE”就表示类型错误，这个改进使得在调试时能更容易地知道遇到的是哪种类型的错误。

3. 整数回复

Redis 虽然没有整数类型，但是却提供了一些用于整数操作的命令，如递增键值的 INCR 命令会以整数形式返回递增后的键值。除此之外，一些其他命令也会返回整数，如可以获取当前数据库中键的数量的 DBSIZE 命令等。整数回复（integer reply）以 (integer) 开头，并在后面跟上整数数据：

```
redis> INCR foo
(integer) 1
```

4. 字符串回复

字符串回复（bulk reply）是最常见的一种回复类型，当请求一个字符串类型键的键值或一个其他类型键中的某个元素时就会得到一个字符串回复。字符串回复以双引号包裹：

```
redis> GET foo
"1"
```

特殊情况是当请求的键值不存在时会得到一个空结果，显示为 (nil)。如：

```
redis> GET noexists
(nil)
```

5. 多行字符串回复

多行字符串回复（multi-bulk reply）同样很常见，如当请求一个非字符串类型键的元素列表时就会收到多行字符串回复。多行字符串回复中的每行字符串都以一个序号开头，如：

```
redis> KEYS *
1) "bar"
2) "foo"
```

> **提示** KEYS 命令的作用是获取数据库中符合指定规则的键名，由于读者的 Redis 中还没有存储数据，所以得到的返回值应该是（empty list or set）。3.1 节会具体介绍 KEYS 命令，此处读者只需了解多行字符串回复的格式即可。

2.4 配置

2.2.1 节中我们通过 redis-server 的启动参数 port 设置了 Redis 的端口号，除此之外 Redis 还支持其他配置选项，如是否开启持久化、日志级别等。由于可以配置的选项较多，通过启动参数设置这些选项并不方便，所以 Redis 支持通过配置文件来设置这些选项。启用配置文件的方法是在启动时将配置文件的路径作为启动参数传递给 redis-server，如：

```
$ redis-server /path/to/redis.conf
```

通过启动参数传递同名的配置选项会覆盖配置文件中相应的参数，就像这样：

```
$ redis-server /path/to/redis.conf --loglevel warning
```

Redis 提供了一个配置文件的模板 redis.conf，位于源代码目录的根目录中。

除此之外还可以在 Redis 运行时通过 CONFIG SET 命令在不重新启动 Redis 的情况下动态修改部分 Redis 配置。就像这样：

```
redis> CONFIG SET loglevel warning
OK
```

并不是所有的配置都可以使用 CONFIG SET 命令修改，附录 B 列出了哪些配置能够使用该命令修改。同样在运行的时候也可以使用 CONFIG GET 命令获得 Redis 当前的配置情况，如：

```
redis> CONFIG GET loglevel
1) "loglevel"
2) "warning"
```

其中第一行字符串回复表示的是选项名，第二行即是选项值。

2.5 多数据库

第 1 章介绍过 Redis 是一个字典结构的存储服务器，而实际上一个 Redis 实例提供了多个用来存储数据的字典，客户端可以指定将数据存储在哪个字典中。这与我们熟知的在

一个关系数据库实例中可以创建多个数据库类似，所以可以将其中的每个字典都理解成一个独立的数据库。

每个数据库对外都是以一个从 0 开始的递增数字命名，Redis 默认支持 16 个数据库，可以通过配置参数 databases 来修改这一数字。客户端与 Redis 建立连接后会自动选择 0 号数据库，不过可以随时使用 SELECT 命令更换数据库，如要选择 1 号数据库：

```
redis> SELECT 1
OK
redis [1]> GET foo
(nil)
```

然而这些以数字命名的数据库又与我们理解的数据库有所区别。首先 Redis 不支持自定义数据库的名字，每个数据库都以编号命名，开发者必须自己记录哪些数据库存储了哪些数据。另外 Redis 也不支持为每个数据库设置不同的访问密码，所以一个客户端要么可以访问全部数据库，要么连一个数据库也没有权限访问。最重要的一点是多个数据库之间并不是完全隔离的，比如 FLUSHALL 命令可以清空一个 Redis 实例中所有数据库中的数据。综上所述，这些数据库更像是一种命名空间，而不适宜存储不同应用程序的数据。比如可以使用 0 号数据库存储某个应用生产环境中的数据，使用 1 号数据库存储测试环境中的数据，但不适宜使用 0 号数据库存储 A 应用的数据而使用 1 号数据库存储 B 应用的数据，不同的应用应该使用不同的 Redis 实例存储数据。由于 Redis 非常轻量级，一个空 Redis 实例占用的内存只有 1MB 左右，所以不用担心多个 Redis 实例会额外占用很多内存。

附录 A

Redis 命令属性

Redis 的不同命令拥有不同的属性，如是否是只读命令，是否是管理员命令等，一个命令可以拥有多个属性。在一些特殊情况下不同属性的命令会有不同的表现，下面来逐一介绍。

A.1 REDIS_CMD_WRITE

拥有 REDIS_CMD_WRITE 属性的命令的表现是会修改 Redis 数据库的数据。一个只读的从数据库会拒绝执行拥有 REDIS_CMD_WRITE 属性的命令；另外在 Lua 脚本中执行了拥有 REDIS_CMD_RANDOM 属性（见 A.4）的命令后，不可以再执行拥有 REDIS_CMD_WRITE 属性的命令，否则会提示错误：“`Write commands not allowed after non deterministic commands.`”

拥有 REDIS_CMD_WRITE 属性的命令如下：

```
SET
SETNX
SETEX
PSETEX
APPEND
DEL
SETBIT
SETRANGE
INCR
DECR
RPUSH
LPUSH
```

RPUSHX
LPUSHX
LINSERT
RPOP
LPOP
BRPOP
BRPOPLPUSH
BLPOP
LSET
LTRIM
LREM
RPOPLPUSH
SADD
SREM
SMOVE
SPOP
SINTERSTORE
SUNIONSTORE
SDIFFSTORE
ZADD
ZINCRBY
ZREM
ZREMRANGEBYSCORE
ZREMRANGEBYRANK
ZUNIONSTORE
ZINTERSTORE
HSET
HSETNX
HMSET
HINCRBY
HINCRBYFLOAT
HDEL
INCRBY
DECRBY
INCRBYFLOAT
GETSET
MSET
MSETNX
MOVE
RENAME
RENAMENX
EXPIRE
EXPIREAT
PEXPIRE
PEXPIREAT
FLUSHDB
FLUSHALL

```
SORT
PERSIST
RESTORE
MIGRATE
BITOP
```

A.2 REDIS_CMD_DENYOOM

拥有 REDIS_CMD_DENYOOM 属性的命令有可能增加 Redis 占用的存储空间，显然拥有该属性的命令都拥有 REDIS_CMD_WRITE 属性，但反之则不然。例如，DEL 命令拥有 REDIS_ CMD_WRITE 属性，但其总是会减少数据库的占用空间，所以不拥有 REDIS_CMD_DENYOOM 属性。

当数据库占用的空间达到了配置文件中 maxmemory 参数指定的值且根据 maxmemory-policy 参数的空间释放规则无法释放空间时，Redis 会拒绝执行拥有 REDIS_CMD_DENYOOM 属性的命令。

> **提示** 拥有 REDIS_CMD_DENYOOM 属性的命令每次调用时不一定都会使数据库的占用空间增大，只是有可能而已。例如，SET 命令当新值长度小于旧值时反而会减少数据库的占用空间。但无论如何，当数据库占用空间超过限制时，Redis 都会拒绝执行拥有 REDIS_CMD_DENYOOM 属性的命令，而不会分析其实际上是不是会真的增加空间占用。

拥有 REDIS_CMD_DENYOOM 属性的命令如下：

```
SET
SETNX
SETEX
PSETEX
APPEND
SETBIT
SETRANGE
INCR
DECR
RPUSH
LPUSH
RPUSHX
LPUSHX
LINSERT
BRPOPLPUSH
LSET
RPOPLPUSH
```

```
SADD
SINTERSTORE
SUNIONSTORE
SDIFFSTORE
ZADD
ZINCRBY
ZUNIONSTORE
ZINTERSTORE
HSET
HSETNX
HMSET
HINCRBY
HINCRBYFLOAT
INCRBY
DECRBY
INCRBYFLOAT
GETSET
MSET
MSETNX
SORT
RESTORE
BITOP
```

A.3　REDIS_CMD_NOSCRIPT

拥有 REDIS_CMD_NOSCRIPT 属性的命令无法在 Redis 脚本中执行。

> **提示**　EVAL 和 EVALSHA 命令也拥有该属性，所以在脚本中无法调用这两个命令，即不能在脚本中调用脚本。

拥有 REDIS_CMD_NOSCRIPT 属性的命令如下：

```
BRPOP
BRPOPLPUSH
BLPOP
SPOP
AUTH
SAVE
MULTI
EXEC
DISCARD
SYNC
REPLCONF
MONITOR
```

```
SLAVEOF
DEBUG
SUBSCRIBE
UNSUBSCRIBE
PSUBSCRIBE
PUNSUBSCRIBE
WATCH
UNWATCH
EVAL
EVALSHA
SCRIPT
```

A.4 REDIS_CMD_RANDOM

当一个脚本执行了拥有 REDIS_CMD_RANDOM 属性的命令后，就不能执行拥有 REDIS_CMD_WRITE 属性的命令了（见 6.4.2 节介绍）。

拥有 REDIS_CMD_RANDOM 的命令如下：

```
SPOP
SRANDMEMBER
RANDOMKEY
TIME
```

A.5 REDIS_CMD_SORT_FOR_SCRIPT

拥有 REDIS_CMD_SORT_FOR_SCRIPT 属性的命令会产生随机结果（见 6.4.2 节），在脚本中调用这些命令时 Redis 会对结果进行排序。

拥有 REDIS_CMD_SORT_FOR_SCRIPT 属性的命令如下：

```
SINTER
SUNION
SDIFF
SMEMBERS
HKEYS
HVALS
KEYS
```

A.6 REDIS_CMD_LOADING

当 Redis 正在启动时（将数据从硬盘载入到内存中），Redis 只会执行拥有 REDIS_CMD_LOADING 属性的命令。

拥有 REDIS_CMD_LOADING 属性的命令如下：

```
INFO
SUBSCRIBE
UNSUBSCRIBE
PSUBSCRIBE
PUNSUBSCRIBE
PUBLISH
```

2.6.11 版本加入了 AUTH，2.6.12 版本加入了 SELECT。

附录 B

配置参数索引

本附录列出了 Redis 中部分配置参数的章节索引，具体见表 B-1。

表 B-1　Redis 部分配置参数列表及章节索引

参　数　名	默　认　值	使用 CONFIG SET 设置	章　　节
daemonize	no	不可以	2.2.1
pidfile	/var/run/redis /pid	不可以	2.2.1
port	6379	不可以	2.2.1
databases	16	不可以	2.5
save	save 900 1 save 300 10 save 60 10000	可以	7.1.1
rdbcompression	yes	可以	7.1.2
rdbchecksum	yes	可以	
dbfilename	dump.rdb	可以	7.1.1
dir	./	不可以	7.1.1
slaveof	无	不可以	8.1.1
masterauth	无	可以	9.1.2
slave-serve-stale-data	yes	可以	8.1.2
slave-read-only	yes	可以	8.1.1
requirepass	无	可以	9.1.2

续表

参 数 名	默 认 值	使用 CONFIG SET 设置	章 节
rename-command	无	不可以	9.1.3
maxmemory	无	可以	4.2.4
maxmemory-policy	volatile-lru	可以	4.2.4
maxmemory-samples	3	可以	4.2.4
appendonly	no	可以	7.1.2
appendfsync	everysec	可以	7.1.2
auto-aof-rewrite-percentage	100	可以	7.1.2
auto-aof-rewrite-min-size	64mb	可以	7.1.2
lua-time-limit	5000	可以	6.4.4
slowlog-log-slower-than	10000	可以	9.3.1
slowlog-max-len	128	可以	9.3.1
hash-max-ziplist-entries	512	可以	4.6.2
hash-max-ziplist-value	64	可以	4.6.2
list-max-ziplist-entries	512	可以	4.6.2
list-max-ziplist-value	64	可以	4.6.2
set-max-intset-entries	512	可以	4.6.2
zset-max-ziplist-entries	128	可以	4.6.2
zset-max-ziplist-value	64	可以	4.6.2

附录 C

CRC16 实现参考

Redis 集群使用 CRC16 对键名进行散列计算来确定键与 slot 的对应关系，其 ANSI C 的实现如下附代码，开发支持 Cluster 特性的客户端时可以以此为参考。该代码摘自 Redis 官方网站（http://redis.io）。

```
/*
 * Copyright 2001-2010 Georges Menie (www.menie.org)
 * Copyright 2010 Salvatore Sanfilippo (adapted to Redis coding style)
 * All rights reserved.
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *     * Redistributions in binary form must reproduce the above copyright
 *       notice, this list of conditions and the following disclaimer in the
 *       documentation and/or other materials provided with the distribution.
 *     * Neither the name of the University of California, Berkeley nor the
 *       names of its contributors may be used to endorse or promote products
 *       derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE REGENTS AND CONTRIBUTORS BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

```
 */

/* CRC16 implementation according to CCITT standards.
 *
 * Note by @antirez: this is actually the XMODEM CRC 16 algorithm, using the
 * following parameters:
 *
 * Name                  : "XMODEM", also known as "ZMODEM", "CRC-16/ACORN"
 * Width                 : 16 bit
 * Poly                  : 1021 (That is actually x^16 + x^12 + x^5 + 1)
 * Initialization          : 0000
 * Reflect Input byte      : False
 * Reflect Output CRC      : False
 * Xor constant to output CRC : 0000
 * Output for "123456789"    : 31C3
 */

static const uint16_t crc16tab[256]= {
    0x0000,0x1021,0x2042,0x3063,0x4084,0x50a5,0x60c6,0x70e7,
    0x8108,0x9129,0xa14a,0xb16b,0xc18c,0xd1ad,0xe1ce,0xf1ef,
    0x1231,0x0210,0x3273,0x2252,0x52b5,0x4294,0x72f7,0x62d6,
    0x9339,0x8318,0xb37b,0xa35a,0xd3bd,0xc39c,0xf3ff,0xe3de,
    0x2462,0x3443,0x0420,0x1401,0x64e6,0x74c7,0x44a4,0x5485,
    0xa56a,0xb54b,0x8528,0x9509,0xe5ee,0xf5cf,0xc5ac,0xd58d,
    0x3653,0x2672,0x1611,0x0630,0x76d7,0x66f6,0x5695,0x46b4,
    0xb75b,0xa77a,0x9719,0x8738,0xf7df,0xe7fe,0xd79d,0xc7bc,
    0x48c4,0x58e5,0x6886,0x78a7,0x0840,0x1861,0x2802,0x3823,
    0xc9cc,0xd9ed,0xe98e,0xf9af,0x8948,0x9969,0xa90a,0xb92b,
    0x5af5,0x4ad4,0x7ab7,0x6a96,0x1a71,0x0a50,0x3a33,0x2a12,
    0xdbfd,0xcbdc,0xfbbf,0xeb9e,0x9b79,0x8b58,0xbb3b,0xab1a,
    0x6ca6,0x7c87,0x4ce4,0x5cc5,0x2c22,0x3c03,0x0c60,0x1c41,
    0xedae,0xfd8f,0xcdec,0xddcd,0xad2a,0xbd0b,0x8d68,0x9d49,
    0x7e97,0x6eb6,0x5ed5,0x4ef4,0x3e13,0x2e32,0x1e51,0x0e70,
    0xff9f,0xefbe,0xdfdd,0xcffc,0xbf1b,0xaf3a,0x9f59,0x8f78,
    0x9188,0x81a9,0xb1ca,0xa1eb,0xd10c,0xc12d,0xf14e,0xe16f,
    0x1080,0x00a1,0x30c2,0x20e3,0x5004,0x4025,0x7046,0x6067,
    0x83b9,0x9398,0xa3fb,0xb3da,0xc33d,0xd31c,0xe37f,0xf35e,
    0x02b1,0x1290,0x22f3,0x32d2,0x4235,0x5214,0x6277,0x7256,
    0xb5ea,0xa5cb,0x95a8,0x8589,0xf56e,0xe54f,0xd52c,0xc50d,
    0x34e2,0x24c3,0x14a0,0x0481,0x7466,0x6447,0x5424,0x4405,
    0xa7db,0xb7fa,0x8799,0x97b8,0xe75f,0xf77e,0xc71d,0xd73c,
    0x26d3,0x36f2,0x0691,0x16b0,0x6657,0x7676,0x4615,0x5634,
    0xd94c,0xc96d,0xf90e,0xe92f,0x99c8,0x89e9,0xb98a,0xa9ab,
    0x5844,0x4865,0x7806,0x6827,0x18c0,0x08e1,0x3882,0x28a3,
    0xcb7d,0xdb5c,0xeb3f,0xfb1e,0x8bf9,0x9bd8,0xabbb,0xbb9a,
    0x4a75,0x5a54,0x6a37,0x7a16,0x0af1,0x1ad0,0x2ab3,0x3a92,
    0xfd2e,0xed0f,0xdd6c,0xcd4d,0xbdaa,0xad8b,0x9de8,0x8dc9,
    0x7c26,0x6c07,0x5c64,0x4c45,0x3ca2,0x2c83,0x1ce0,0x0cc1,
    0xef1f,0xff3e,0xcf5d,0xdf7c,0xaf9b,0xbfba,0x8fd9,0x9ff8,
```

```
    0x6e17,0x7e36,0x4e55,0x5e74,0x2e93,0x3eb2,0x0ed1,0x1ef0
};

uint16_t crc16(const char *buf, int len) {
    int counter;
    uint16_t crc = 0;
    for (counter = 0; counter < len; counter++)
           crc = (crc<<8) ^ crc16tab[((crc>>8) ^ *buf++)&0x00FF];
    return crc;
}
```