

# Rcpp介绍与应用

**Jiayue Zeng**

May 18, 2018

# Outline

- Rcpp介绍
- Rcpp练习
- R包结构介绍
- 构建R包练习
- 一些小栗子

# 1 Rcpp 包

## 1.1 Rcpp R与C++的无缝整合

- 作者:
- Dirk Eddelbuettel
- Romain Francois, JJ Allaire, Kevin Ushey, Qiang Kou, Nathan Russell, Douglas Bates and John Chambers
- Seamless R and C++ Integration with Rcpp (Springer, 2013)
- Rcpp Gallery: [gallery.rcpp.org/](http://gallery.rcpp.org/)

# 1 Rcpp 包

## 1.1 Rcpp R与C++的无缝整合

- `install.packages("Rcpp")`

- C++ 编译器

Windows: 安装Rtools

Mac: 安装Xcode

Linux: 使用 `sudo apt-get install r-base-dev`

# 1 Rcpp 包

## 1.2 cppFunction( )

- cppFunction 允许我们在R中编写c++函数

```
library(Rcpp)
cppFunction(
  'int fib_cpp_0(int n){

  }'
)
```

# 1 Rcpp 包

## 1.2 cppFunction( )

- cppFunction 允许我们在R中编写c++函数

```
library(Rcpp)
cppFunction(
  'int fib_cpp_0(int n){
    if(n==1||n==2) return 1;
    return(fib_cpp_0(n-1)+fib_cpp_0(n-2));
  }'
)
```

# 1 Rcpp 包

## 1.2 cppFunction( )

- fib\_cpp\_0与fib\_r对比

```
fib_r <- function(n){  
  if(n==1||n==2) return(1)  
  return(fib_r(n-1)+fib_r(n-2))  
}
```

# 1 Rcpp 包

## 1.2 cppFunction( )

- fib\_cpp\_0与fib\_r对比

```
> system.time(fib_r(30))  
用户 系统 流逝  
0.78 0.00 0.80  
> system.time(fib_cpp_0(30))  
用户 系统 流逝  
0.02 0.00 0.01
```



# 1 Rcpp 包

## 1.3 sourceCpp( )

- C++ 文件

```
#include <Rcpp.h>  
using namespace Rcpp;
```

# 1 Rcpp 包

## 1.3 sourceCpp( )

- C++ 文件

```
#include <Rcpp.h>  
using namespace Rcpp;  
  
//[[Rcpp::export]]
```

# 1 Rcpp 包

## 1.3 sourceCpp( )

- C++ 文件

```
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
int fib_cpp_1(int n)
{

}
```

# 1 Rcpp 包

## 1.3 sourceCpp( )

- C++ 文件

```
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
int fib_cpp_1(int n)
{
    if(n==1||n==2) return 1;
    return fib_cpp_1(n-1)+fib_cpp_1(n-2);
}
```

# 1 Rcpp 包

## 1.3 sourceCpp( )

- 使用sourceCpp
- 它将创建匹配的R函数并将它们添加到当前的会话中

```
setwd("...")  
sourceCpp("fib_cpp_1.cpp")
```

- 注意这些函数不能保存到.Rdata文件中
- 每次重启R时必须重新创建它们

# 1 Rcpp 包

## 1.3 sourceCpp( )

- fib\_cpp\_0与fib\_cpp\_1对比

```
> system.time(fib_cpp_0(50))  
用户 系统 流逝  
26.31 0.00 26.60  
> system.time(fib_cpp_1(50))  
用户 系统 流逝  
26.09 0.00 26.27
```

# 1 Rcpp 包

## 1.3 sourceCpp( )

- .Call

```
> fib_cpp_1  
function (n)  
  .Call(<pointer: 0x0000000063c015d0>, n)
```

## 2 创建R包

将标准的C++源文件打包有以下几点好处：

- 没有C++开发工具的用户也可以使用你的代码；
- R软件包构建系统可以将多个源文件以及它依赖的文件打包在一起；
- 软件包为测试、文档和一致性提供了附加的基础设施；



## 2 创建R包

### 2.1 package.skeleton & Rcpp.package.skeleton

- 使用Rcpp.package.skeleton()函数至今仍是最简单的办法
- 1、它生成了一个扩展包所需的所有文件
- 2、它同时包括了使用Rcpp所需的不同组成部分

```
> Rcpp.package.skeleton("RcppDemo")
```

## 2 创建R包

### 2.1 package.skeleton & Rcpp.package.skeleton

```
> Rcpp.package.skeleton("RcppDemo")
```

```
Creating directories ...
```

```
Creating DESCRIPTION ...
```

```
Creating NAMESPACE ...
```

```
Creating Read-and-delete-me ...
```

```
Saving functions and data ...
```

```
Making help files ...
```

```
Done.
```

```
Further steps are described in './RcppDemo/Read-and-delete-me'.
```

```
Adding Rcpp settings
```

```
>> added Imports: Rcpp
```

```
>> added LinkingTo: Rcpp
```

```
>> added useDynLib directive to NAMESPACE
```

```
>> added importFrom(Rcpp, evalCpp) directive to NAMESPACE
```

```
>> added example src file using Rcpp attributes
```

```
>> added Rd file for rcpp_hello_world
```

```
>> compiled Rcpp attributes
```

## 2 创建R包

### 2.1 package.skeleton & Rcpp.package.skeleton

- 使用Rcpp.package.skeleton()函数至今仍是最简单的办法
- 1、它生成了一个扩展包所需的所有文件
- 2、它同时包括了使用Rcpp所需的不同组成部分


```
Rcpp.package.skeleton("NewPakcage",  
                      example_code = FALSE,  
                      cpp_files = c('fib_cpp_1.cpp'))
```

## 2 创建R包


### 2.1 package.skeleton & Rcpp.package.skeleton


- 使用Rcpp.package.skeleton()函数至今仍是最简单的办法
- 1、它生成了一个扩展包所需的所有文件
- 2、它同时包括了使用Rcpp所需的不同组成部分


☐ 名称


 man

 R

 src

 DESCRIPTION

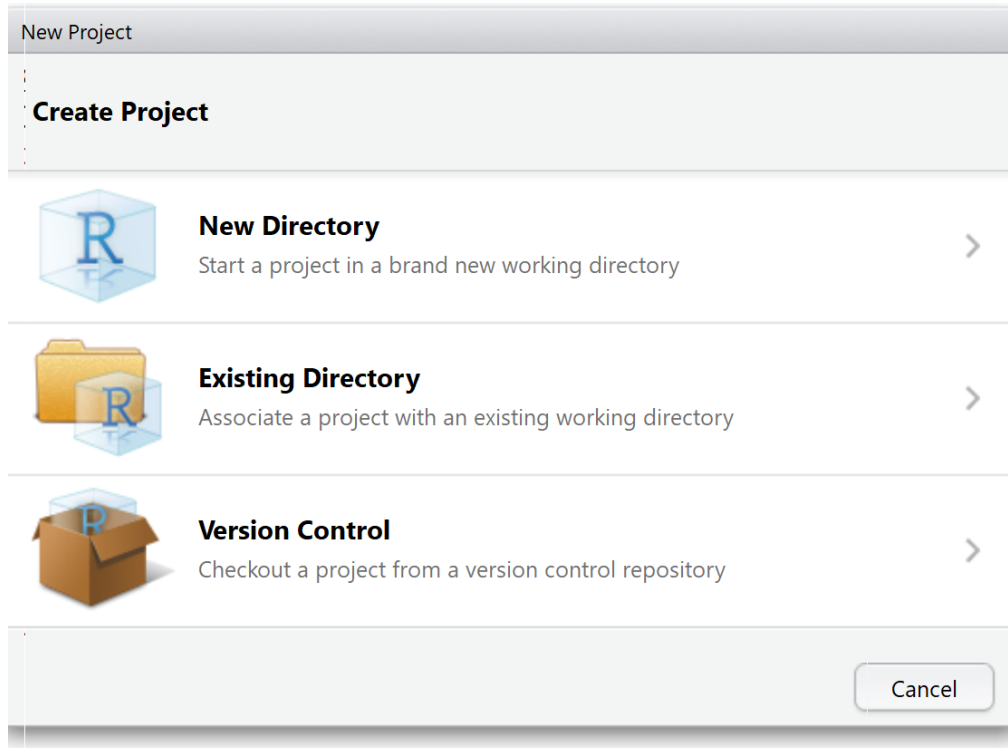
 NAMESPACE

 Read-and-delete-me

## 2 Build a package

### 2.2 Rstudio

- Creat Project 也很方便



## 2 创建R包

### 2.3 DESCRIPTION

- Package: RcppDemo
- Type: Package
- Title: What the Package Does in One 'Title Case' Line
- Version: 1.0
- Date: 2018-05-18
- Author: Your Name
- Maintainer: Your Name <your@email.com>
- Description: One paragraph description of what the package does as one or
  - more full sentences.
- License: GPL ( $\geq 2$ )
- Imports: Rcpp ( $\geq 0.12.16$ )
- LinkingTo: Rcpp

## 2 创建R包

### 2.3 DESCRIPTION

- Package: RcppDemo
- Type: Package
- Title: What the Package Does in One 'Title Case' Line
- Version: 1.0
- Date: 2018-05-18
- Author: Your Name
- Maintainer: Your Name <your@email.com>
- Description: One paragraph description of what the package does as one or
  - more full sentences.
- License: GPL ( $\geq 2$ )
- Imports: Rcpp ( $\geq 0.12.16$ )
- LinkingTo: Rcpp

## 2 创建R包

### 2.4 NAMESPACE

- `importFrom(Rcpp,sourceCpp)`
- `importFrom(Rcpp, evalCpp)`



## 2 创建R包

### 2.4 NAMESPACE

```
useDynLib(RcppDemo, .registration=TRUE)  
exportPattern("^[:alpha:]+")  
importFrom(Rcpp, evalCpp)
```

## 2 创建R包

### 2.4 NAMESPACE

```
# Generated by roxygen2: do not edit by hand
```

```
export(dsplitn)  
export(dsplitt)  
export(psplitn)  
export(psplitt)  
export(qsplitn)  
export(qsplitt)  
export(rsplitn)  
export(rsplitt)  
importFrom(Rcpp,sourceCpp)  
importFrom(stats,qnorm)  
useDynLib(dng)
```

## 2 创建R包

### 2.4 NAMESPACE

- 1、其确保有Rcpp.package.skeleton( )生成的动态链接库会被载入从而能被新生成的R扩展包使用。
- 2、其声明了在这个扩展包的命名空间中全局可见的函数或数据集。
- 在默认情况下，我们通过正则表达式将所有以字母开头的函数全部导出

## 2 创建R包

### 2.5 help 文档

```
\name{rcpp_hello_world}  
\alias{rcpp_hello_world}  
\docType{package}  
\title{  
Simple function using Rcpp  
}  
\description{  
Simple function using Rcpp  
}  
\usage{  
rcpp_hello_world()  
}  
\examples{  
\dontrun{  
rcpp_hello_world()  
}  
}
```

## 2 创建R包

### 2.5 help 文档

```
\name{rcpp_hello_world}  
\alias{rcpp_hello_world}  
\docType{package}  
\title{  
Simple function using Rcpp  
}  
\description{  
Simple function using Rcpp  
}  
\usage{  
rcpp_hello_world()  
}  
\examples{  
\dontrun{  
rcpp_hello_world()  
}  
}
```

```
\name{ }  
\alias{ }  
\title{ }  
\description{ }  
\usage{ }  
\arguments{  
  \item{x}{ }  
  \item{y}{ }  
}  
\details{ }  
\value{ }  
\references{ }  
\author{ }  
\seealso{ }  
\examples{ }
```

## 2 创建R包

### 2.6 其他

- CITATION 引用
- NEWS 版本

Changes in version 0.2.0 (2018-03-15)

- o Include split normal distribution and related moment functions based on Villani and Larsson (2006).

Changes in version 0.1.1 (2017-03-02)

- o Include split-t distribution and related moment functions based on Li, Villani and Kohn (2010).

## 2 创建R包

### 2.6 Build and Check

- R CMD build
- R CMD check
- devtools::document()
- Rcpp::compileAttributes
- Test that

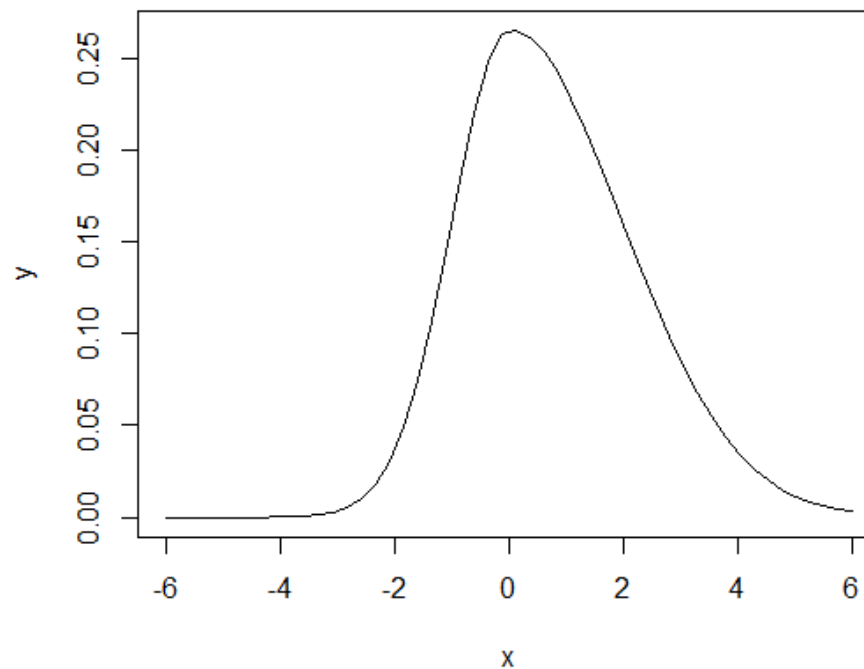
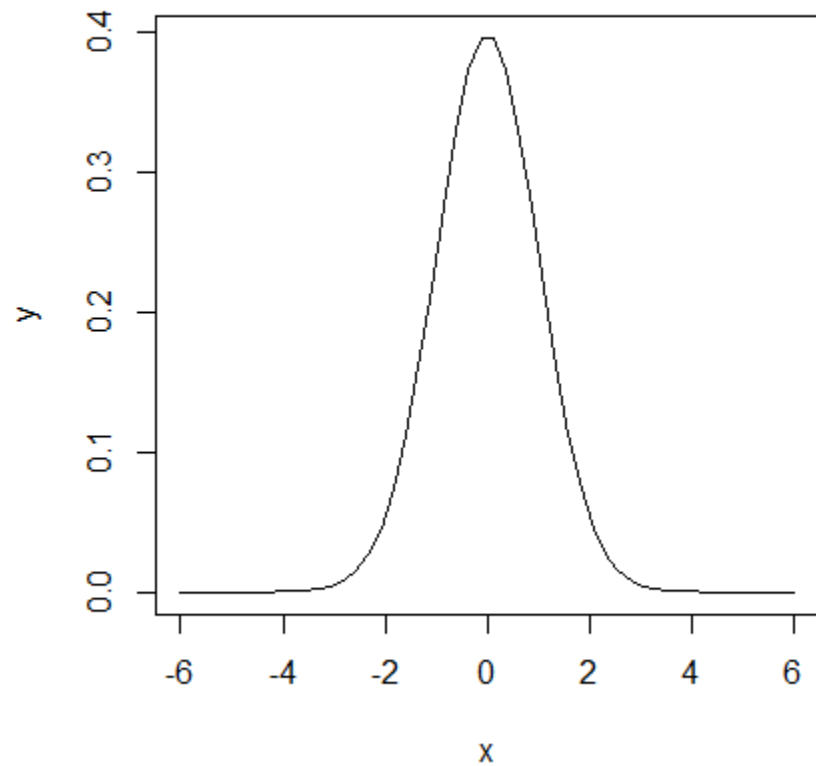
## 2 创建R包

### 2.7 例子 dng

- split normal

`dsplitn(x, mu=0, sigma=1, lmd=1)`

`dsplitn(x, mu=0, sigma=1, lmd=2)`





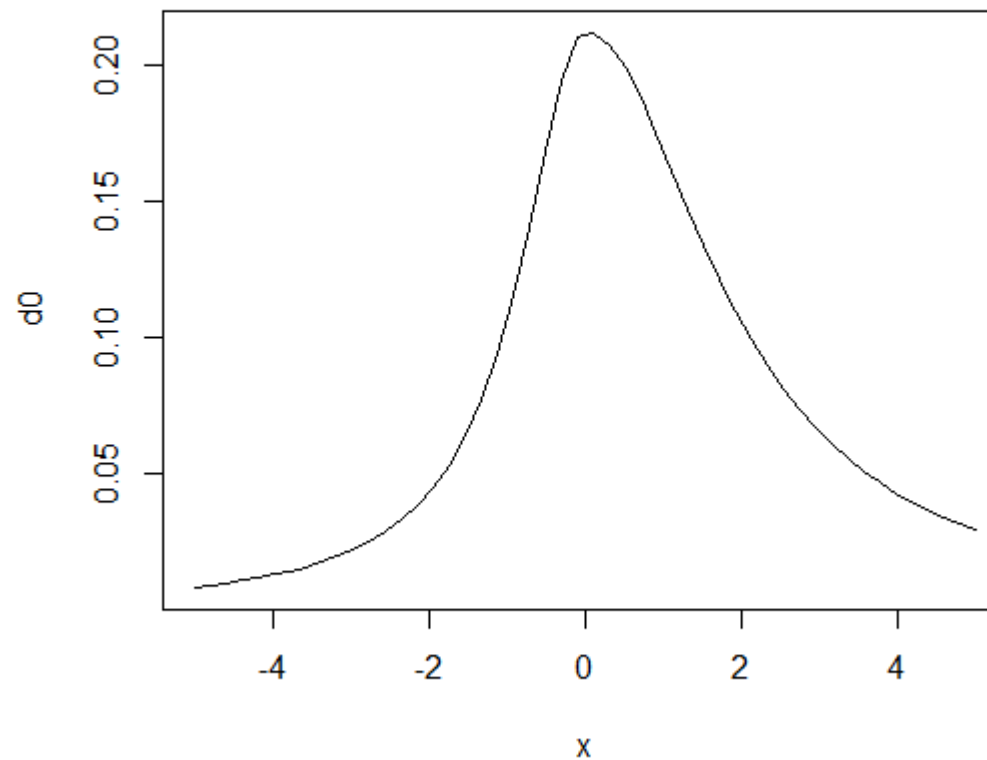
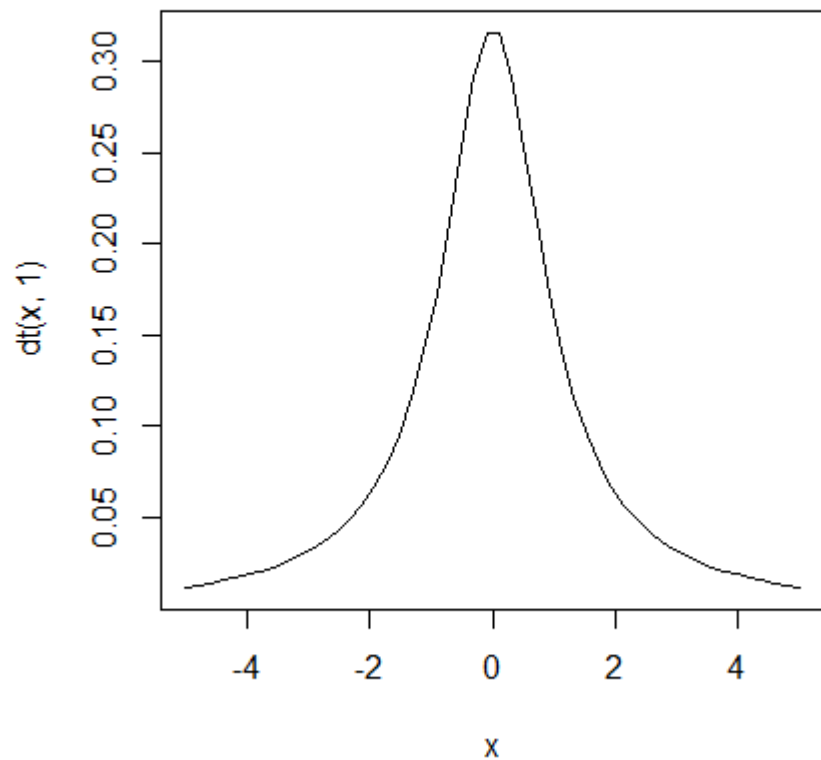
## 2 创建R包

### 2.7 例子 dng

- split t

`dsplitt(x, mu=0, df=1, phi=1, lmd=1, log = FALSE)`

`dsplitt(x, mu=0, df=1, phi=1, lmd=2, log = FALSE)`



## 3 数据类型

### 3.1 常用数据类型

- Double  $\Leftrightarrow$  numeric
- Int  $\Leftrightarrow$  integer
- String  $\Leftrightarrow$  character
- Logical  $\Leftrightarrow$  bool

## 3 数据类型

### 3.1 常用数据类型

关键字	描述
int/double/bool/String/auto	整数型/数值型/布尔值/字符型/自动识别(C++11)
IntegerVector	整型向量
NumericVector	数值型向量(元素的类型为double)
ComplexVector	复数向量 Not Sure
LogicalVector	逻辑型向量； R的逻辑型变量可以取三种值：TRUE, FALSE, NA； 而C++布尔值只有两个,true or false。如果将R的NA转化为C++中的布尔值，则会返回true。
CharacterVector	字符型向量
ExpressionVector	vectors of expression types
RawVector	vectors of type raw
IntegerMatrix	整型矩阵
NumericMatrix	数值型矩阵(元素的类型为double)
LogicalMatrix	逻辑型矩阵
CharacterMatrix	字符矩阵
List/ GenericVector	列表； 类似于R中列表， 其元素可以使任何数据类型
DataFrame	数据框； data frames； 在Rcpp内部， 数据框其实是通过列表实现的
Function	函数型
Environment	环境型； 可用于引用R环境中的函数、其他R包中的函数、操作R环境中的变量
RObject	可以被R识别的类型

## 3 数据类型

### 3.1 常用数据类型

- Robject类
- 它是构建Rcpp API的基础类
- 每个Robject类实例都封装了一个R对象
- 每个R对象都可以在内部表示为一个SEXP
- 依赖控制域的特定值可以用来表示不同的类型
- 给定一个联合类型，根据控制域的值，剩余的字节会被解析成控制域所暗示的类型。

## 3 数据类型

### 3.1 常用数据类型

- Robject类
- SEXP对象被认为是不透明的
- 只能通过辅助函数间接访问和查看
- 很多用户常见的类都基于Robject类

## 3 数据类型

### 3.2 数据类型的建立

```
//1. Vector
NumericVector V1(n);
NumericVector V2=NumericVector::create(1, 2, 3);
LogicalVector V3=LogicalVector::create(true,false,R_NaN);
//2. Matrix
NumericMatrix M1(nrow,ncol);
//3. Multidimensional Array
NumericVector out=NumericVector(Dimension(2,2,3));
```

## 3 数据类型

### 3.2 数据类型的建立

//4. List

```
NumericMatrix y1(2,2);
```

```
NumericVector y2(5);
```

```
List L=List::create(Named("y1")=y1,  
                    Named("y2")=y2);
```

//5. DataFrame

```
NumericVector a = NumericVector::create(1,2,3);
```

```
CharacterVector b = CharacterVector::create("a","b","c");
```

```
std::vector<std::string> c(3);
```

```
c[0]="A"; c[1]="B"; c[2]="C";
```

```
DataFrame DF=DataFrame::create(Named("col1")=a,  
                                Named("col2")=b,  
                                Named("col3")=c);
```

## 4 调用R函数

### 4.1 Function类型

- 将R函数放入Function类型的对象中。这样就从C++中直接调用R函数



## 4 调用R函数

### 4.1 Function类型

- 将R函数放入Function类型的对象中。这样就从C++中直接调用R函数

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
RObject callwithone(Function f) {
    return f(1);
}
```

## 4 调用R函数

### 4.1 Function类型

- 将R函数放入Function类型的对象中。这样就从C++中直接调用R函数

```
sourceCpp("callwithone.cpp")  
> callwithone(paste)  
[1] "1"  
> callwithone(function(x) x+1)  
[1] 2
```

## 4 调用R函数

### 4.2 语法糖

- 算数和逻辑运算符
- 逻辑总结函数
- 向量视图
- 其他有用的函数

## 4 调用R函数

### 4.2 语法糖

算数和逻辑运算符:

- 所有的基本算数和逻辑运算符都是向量化的:
- $+$ ,  $-$ ,  $*$ ,  $/$ ,
- $++$ ,  $--$ ,
- $\text{pow}(x,p)$ ,
- $<$ ,  $<=$ ,  $>$ ,  $>=$ ,
- $==$ ,  $!=$

## 4 调用R函数

### 4.2 语法糖

逻辑总结函数:

- 语法糖函数`any()`和`all()`都是完全惰性的
- 例如`any(x==0)`, 可能只需要对向量中的一个元素进行评估, 并返回一个可以被`.is_true()`, `.is_false()`, `.is_na()`转换成bool的特殊类型

## 4 调用R函数

### 4.2 语法糖

向量视图:

- `head( )`, `tail( )`,
- `rep_each( )`, `rep_len( )`, `rev( )`
- `seq_along( )`, `seq_len( )`

## 4 调用R函数

### 4.2 语法糖

向量视图：

- `head()` , `tail()` ,
  - `rep_each()` , `rep_len()` , `rev()`
  - `seq_along()` , `seq_len()`
- 
- 在R中这些函数在执行时都对向量进行复制
  - 但是在Rcpp中他们都是简单的指向原向量，覆盖子集选取操作而实现的特殊行为，所以他们的效率很高。
- 
- Eg: `rep_len(x, 1e6)`

## 4 调用R函数

### 4.2 语法糖

其他有用的函数：

- 数学函数：

`abs()`, `acos()`, `asin()`, `atan()`, `beta()`, `ceil()`, `ceiling()`, `choose()`, `cos()`, `cosh()`, `digamma()`, `exp()`, `expm1()`, `factorial()`, `floor()`, `gamma()`, `lbeta()`, `lchoose()`, `lfactorial()`, `lgamma()`, `log()`, `log10()`, `log1p()`, `pentagamma()`, `psigamma()`, `round()`, `signif()`, `sin()`, `sinh()`, `sqrt()`, `tan()`, `tanh()`, `tetragamma()`, `trigamma()`, `trunc()`.

- 汇总函数: `mean()`, `min()`, `max()`, `sum()`, `sd()`, and (for vectors) `var()`
- 返回向量的汇总函数: `cumsum()`, `diff()`, `pmin()`, and `pmax()`
- 查找函数: `match()`, `self_match()`, `which_max()`, `which_min()`
- 重复值处理函数: `duplicated()`, `unique()`
- 所有标准分布的: d/p/q/r 开始的函数



## 4 调用R函数

### 4.3 其他

- R::
- Import其他R包
- 自己写一下

## 5 STL

- STL: 标准模板库
- Boost: [www.boost.org/doc/](http://www.boost.org/doc/)
- 安装boost都在文件的开头加上特定的头文件
- `#include <boost/array.hpp>`

## 5 STL

### 5.1 数据结构

- STL提供了大量的数据结构:
- Array, bitset,
- map,multimap,unordered\_map, unordered\_multimap
- set, multiset, unordered\_set, unordered\_multiset

## 5 STL

### 5.2 使用迭代器

- 迭代器主要包含3个主要的运算符： ++, \*, ==

```
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
double sum3(NumericVector x) {
    double total = 0;
    NumericVector::iterator it;
    for(it = x.begin(); it != x.end(); ++it) {
        total += *it;
    }
    return total;
}
```

## 5 STL

### 5.3 算法

- 头文件<algorithm>中提供了许多的算法

```
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
IntegerVector findInterval2(NumericVector x, NumericVector breaks) {
    IntegerVector out(x.size());
    NumericVector::iterator it, pos;
    IntegerVector::iterator out_it;
    for(it = x.begin(), out_it = out.begin(); it != x.end();
        ++it, ++out_it) {
        pos = std::upper_bound(breaks.begin(), breaks.end(), *it);
        *out_it = std::distance(breaks.begin(), pos);
    }
    return out;
}
```

# Rcpp介绍与应用

谢谢大家！