代码工作流总览

提供的代码文件来自于 RTKLIB 库，这是一个用于标准和精密 GNSS 定位的开源程序包。

1. 入口点 (rtknavi.c):

rtknavi.c 文件中的 main() 函数是整个程序的入口。它的主要作用是初始化并启动一个 RTK 服务（rtksvrstart）。它负责配置处理选项（prcopt_t）、解算选项（solopt_t）以及设置数据输入/输出流（例如，从 RTCM3 文件读取流动站和基站数据，并将解算结果输出到日志文件）。

2. 核心定位流程 (pntpos.c):

main() 函数启动服务后，核心的定位解算（在您提供的文件中）主要由 pntpos.c 文件中的 pntpos() 函数执行。这是一个**单点定位（SPP）**函数，它使用伪距和多普勒观测值来计算接收机的位置、速度和钟差（PVT）。

3. **辅助模块**:
    - ephemeris.c 和 preceph.c：用于计算卫星的位置、速度和钟差。
    - rcvraw.c 和 rtcm3.c：用于解码导航电文（如星历、电离层参数）。
    - ionex.c 和 geoid.c：用于电离层和大地水准面改正。
    - lambda.c：用于 RTK 模式下的模糊度固定（在 pntpos 流程中不使用，但在 rtknavi 配置的 RTK 模式中至关重要）。
    - rtcm.c: RTCM 消息的通用入口，它会调用 rtcm3.c 中的函数。

分函数详细流程

我们将以 pntpos() 函数作为核心工作流进行追踪，这是计算一个历元（epoch）解的具体实现。

1. rtknavi.c: main() - 程序入口与服务启动

- **初始值**:
    - prcopt_t PrcOpt：定义了核心处理选项。例如 PMODE_KINEMA（动态模式）、navsys=51（GPS+GLO+GAL）、ionoopt=IONOOPT_BRDC（广播星历模型）。
    - solopt_t SolOpt：定义了解算输出选项，如 SOLF_LLH（经纬高格式）。
    - char* paths[]：定义了输入和输出文件路径。

- **处理流程**：
    1. 初始化服务结构体：rtksvrinit(&rtksvr)。
    2. 初始化并打开监控流：strinit(&monistr) 和 stropen(...)。
    3. （可选）打开调试追踪文件：traceopen(TRACEFILE)。
    4. 配置 PrcOpt 结构体：设置流动站（rovpos）和基准站（refpos）的位置类型（例如，从 RTCM 读取基站位置）。
    5. 配置输入输出流：设置哪些流是活动的（strs）以及它们各自的格式（Format）。
    6. **启动 RTK 服务**：调用 rtksvrstart(...)。此函数启动一个或多个线程，在后台循环处理数据。
- **输出**：
    - rtksvrstart 返回一个状态（1 为成功）。真正的"输出"（定位解）是在服务线程中异步生成的，并根据 paths 中的配置写入到日志文件。

2. pntpos.c: pntpos() - 单点定位主函数

此函数在 rtksvr 服务的处理线程中被调用（虽然 rtksvr.c 未提供，但这是 pntpos 的典型用法）。

- **输入**：obs（观测数据数组）、n（观测数量）、nav（导航电文）、opt（处理选项）。
- **输出**：sol（填充完毕的解算结构体）、azel（卫星方位角/仰角）、ssat（卫星状态）、msg（错误信息）。
- **初始值**：
    - sol->stat = SOLQ_NONE（解状态设为"无解"）。
    - 为 rs（6xn）、dts（2xn）、var（1xn）、azel_（2xn）、resp（1xn）分配内存。
- **处理流程**：
    1. **计算卫星状态**：调用 ephemeris.c: satposs()。
    2. **估计接收机位置**：调用 pntpos.c: estpos()。
    3. **完好性监测 (RAIM)**：如果 estpos 失败且 n >= 6 并且 RAIM 选项已开启（opt->posopt[4]），则调用 pntpos.c: raim_fde() 尝试排除故障卫星并重新解算。

4. **估计接收机速度**：如果 estpos 成功（stat=1），则调用 pntpos.c: estvel()。
5. 填充 ssat 和 azel 结构体以供输出。

- **最终输出**：stat（解算状态，1 为成功）和更新后的 sol 结构体。

3. ephemeris.c: satposs() - 计算所有卫星的位置和钟差

- **输入**：teph（星历选择时间）、obs（观测数据）、n、nav、ephopt（星历选项）。
- **输出**：rs（卫星位置/速度数组）、dts（卫星钟差/钟漂数组）、var（方差数组）、svh（健康标志数组）。
- **处理流程**：
  1. 遍历所有 n 个观测数据 obs[i]。
  2. 搜索伪距 pr 并计算信号发射时间 time[i] = obs[i].time - pr / CLIGHT。
  3. **初步钟差**：调用 ephemeris.c: ephclk()，使用广播星历计算初步的卫星钟差 dt。
  4. **修正发射时间**：time[i] = time[i] - dt。
  5. **计算卫星 PVT**：调用 ephemeris.c: satpos()，使用修正后的 time[i] 来获取该卫星的精确位置、速度和最终钟差。
- **最终输出**：填充 rs, dts, var, svh 数组。

4. ephemeris.c: satpos() - 计算单个卫星的位置和钟差

- **输入**：time（修正后的发射时间）、teph、sat、ephopt、nav。
- **输出**：rs（单个卫星的位置/速度）、dts（单个卫星的钟差/钟漂）、var、svh。
- **处理流程**：
  - 这是一个分发函数，根据 ephopt（星历选项）选择计算方法：
  - **情况 1：EPHOPT_BRDC (广播星历)**
    - 调用 ephemeris.c: ephpos()。
    - ephpos 内部：
      1. 调用 seleph (GPS/GAL 等) 或 selgeph (GLO) 选择最合适的广播星历。

2. 调用 ephemeris.c: eph2pos() (GPS/GAL) 或 geph2pos() (GLO) 计算 time 时刻的卫星位置 rs、钟差 dts 和方差 var。
3. **计算速度**：通过增加一个极小的时间 tt（1E-3 秒）并再次调用 eph2pos/geph2pos，然后通过差分近似计算卫星速度 rs[i+3] = (rst[i] - rs[i]) / tt 和钟漂 dts[1]。

- **情况 2：EPHOPT_PREC (精密星历)**
  - 调用 preceph.c: peph2pos()。
  - peph2pos 内部：
    1. 调用 preceph.c: pephpos() 利用 SP3 文件数据和多项式插值计算卫星位置和初步钟差。
    2. 调用 preceph.c: pephclk() 利用 CLK 文件数据和线性插值精炼卫星钟差。
    3. 调用 preceph.c: satantoff() 计算并应用卫星天线相位中心偏移（PCO）改正。
    4. 应用相对论效应改正。
    5. 像 ephpos 一样，通过微小时间 tt 差分来计算速度和钟漂。

- **最终输出**：rs (6 元, pos+vel), dts (2 元, bias+drift), var, svh。

5. pntpos.c: estpos() - 迭代最小二乘估计位置

- **输入**：obs, n, rs, dts, var, svh, nav, opt。
- **输出**：sol（更新位置和钟差）、azel、vsat、resp、msg。
- **处理流程**：
  1. 初始化状态向量 x（3 个位置 + 1 个 GPS 钟差 + 4 个系统间钟差，共 8 个未知数）。
  2. 进入迭代循环（最多 MAXITR 次）：
  3. **计算残差和矩阵**：调用 pntpos.c: rescode()。
     - rescode 内部：

a. 用当前迭代的 x 中的位置 rr 和钟差 dtr，计算接收机的大地坐标 pos。

b. 遍历所有卫星 i:

c. 排除卫星：调用 satexclude()（根据健康位 svh 和 opt）。

d. 计算几何距离：r = geodist(rs+i*6, rr, e)，同时得到视线向量 e。

e. 计算方位角/仰角：satazel(pos, e, azel+i*2)。

f. 筛选：排除低于仰角（opt->elmin）和信噪比（snrmask）门限的卫星。

g. 电离层改正：调用 pntpos.c: ionocorr()，根据 opt->ionoopt：

* IONOOPT_BRDC：调用 ionmodel() (使用 nav->ion_gps 中的 Klobuchar 参数)。

* IONOOPT_TEC：调用 ionex.c: iontec() (使用 nav->tec 中的 IONEX 格网数据)。

h. 对流层改正：调用 pntpos.c: tropcorr()，根据 opt->tropopt：

* TROPOPT_SAAS：调用 tropmodel() (使用 Saastamoinen 模型)。

* TROPOPT_SBAS：调用 sbstropcorr()。

i. 计算改正后的伪距：调用 pntpos.c: prange()。

* prange 内部:

* 获取 P1 和 P2 伪距。

* 应用码偏差改正（DCB）：P1 += nav->cbias[sat-1][1]。

* 应用 TGD（时间群延迟）：P1 -= gettgd(sat, nav, 0)。

* 如果使用双频消电离层（IONOOPT_IFLC），则计算组合伪距 (P2 - gamma*P1) / (1.0 - gamma)。

j. 计算伪距残差：v[nv] = P - (r + dtr - CLIGHT*dts[i*2] + dion + dtrp)。

k. 构建 H 矩阵：H 矩阵的列对应 [dx, dy, dz, dtr_GPS, dtr_GLO, ...].

l. 计算方差：var[nv] = varerr(...) + vare[i] + vmeas + vion + vtrp。

m. 设置 vsat[i] = 1，resp[i] = v[nv]，(*ns)++。

- rescode 返回：nv（有效方程数）。

4. 如果 nv < NX（方程数 < 未知数），则中断并报告 "lack of valid sats"。
5. **加权最小二乘**：使用 var 数组对 v 和 H 进行加权。
6. **求解**：调用 lsq(H, v, NX, nv, dx, Q)（库函数，未提供）解算法方程，得到状态修正量 dx 和协方差 Q。

7. **更新状态**：x[j] += dx[j]。
8. **检查收敛**：如果 norm(dx) < 1E-4，则迭代收敛。

- **最终输出**：
  - 如果收敛：sol->rr = x[0:2]，sol->dtr[0] = x[3]/CLIGHT（GPS 钟差），sol->dtr[1] = x[4]/CLIGHT（GLO-GPS 钟差），以此类推。
  - 调用 pntpos.c: valsol() 检查解的质量（卡方检验和 GDOP）。
  - 返回 stat = 1。

6. pntpos.c: estvel() - 迭代最小二乘估计速度

- **输入**：obs, n, rs, dts, nav, opt, sol（包含 estpos 得到的位置 sol->rr），azel, vsat。
- **输出**：更新 sol->rr+3（速度）和 sol->qv（速度协方差）。
- **处理流程**：

1. 初始化状态向量 x（4 个未知数：vx, vy, vz，钟漂）。
2. 进入迭代循环（最多 MAXITR 次）：
3. **计算多普勒残差**：调用 pntpos.c: resdop()。
   - resdop 内部：

a. 遍历所有 vsat[i] == 1 的卫星。

b. 获取载波频率 freq。

c. 计算卫星和接收机（来自 x）的相对速度 vs。

d. 计算几何距离变化率 rate（包含地球自转改正）。

e. 计算多普勒残差：v[nv] = (-obs[i].D[0]*CLIGHT/freq - (rate + x[3] - CLIGHT*dts[1+i*2])) / sig。

f. 构建 H 矩阵（4 列）。

   - resdop 返回：nv（有效方程数）。

4. 如果 nv < 4，中断。
5. **求解**：调用 lsq(H, v, 4, nv, dx, Q) 解算速度修正量 dx。
6. **更新状态**：x[j] += dx[j]。
7. **检查收敛**：如果 norm(dx) < 1E-6，则迭代收敛。

- **最终输出**：
    - 如果收敛：sol->rr+3 = x[0:2]，并填充速度协方差矩阵 sol->qv。

总结

1. **启动**：rtknavi.c 的 main 函数启动 RTK 服务，配置好输入（RTCM 文件）、处理模式（Kinematic）和输出（日志文件）。
2. **数据处理循环**（由 rtksvrstart 触发，pntpos 是核心）：
3. **获取卫星 PVT**：pntpos() 调用 satposs()，后者循环每颗卫星，通过 ephclk() 修正发射时间，然后调用 satpos() 获取精确的卫星位置、速度和钟差。
   satpos() 根据配置选择使用广播星历（eph2pos/geph2pos）还是精密星历（peph2pos）。
4. **计算接收机位置**：pntpos() 调用 estpos()，后者进入迭代最小二乘循环。
5. **建立观测方程**：在 estpos 循环中，rescode() 函数被调用。它为每颗有效卫星计算：
    - 几何距离 r。
    - 电离层延迟 dion（通过 ionocorr，可能使用 ionex.c）。
    - 对流层延迟 dtrp（通过 tropcorr）。
    - 改正后的伪距 P（通过 prange，应用 TGD 和 DCB）。
    - 计算残差 v 和设计矩阵 H。
6. **求解位置**：estpos 使用 lsq 求解 dx 并更新状态 x，直到收敛。
7. **质量控制**：valsol() 通过 GDOP 和卡方检验来验证解的可靠性。
8. **计算接收机速度**：pntpos() 调用 estvel()，后者在另一次迭代最小二乘循环中调用 resdop()，利用多普勒观测值解算接收机速度和钟漂。
9. **输出**：最终的 sol_t 结构体（包含位置、速度、钟差、钟漂和质量控制信息）被返回，并由 rtknavi 服务写入文件。